Memory Determines Learning Direction: A Theory of Gradient-Based Optimization in State Space Models

JingChuan Guan [*], **Tomoyuki Kubota,** **Yasuo Kuniyoshi,** **Kohei Nakajima**
Laboratory for Intelligent Systems and Informatics
Department of Mechano-Informatics, School of Information Science and Technology,
The University of Tokyo

Abstract

State space models (SSMs) have shown strong potential to outperform Transformers, but their learning mechanisms remain poorly understood, particularly the conditions for successful parameter updates that preserve supervisory information. Prior work attributes the difficulty of such updates to the vanishing gradient problem during backpropagation through time, in which error signals decay exponentially through repeated gradient multiplication. In this work, by expressing error signals explicitly as functions of inputs, we show that the conventional vanishing gradient framework is insufficient to explain a loss of supervisory information. Using this representation of error, we reveal that the memory of the input encoded in supervisory information does not necessarily decay exponentially, and we analytically demonstrate that the loss of supervisory information cannot be alleviated through training. This result theoretically confirms the importance of the initial weights of SSMs and suggests that the recurrent layer does not require training. By performing experiments on both linear SSMs and a representative nonlinear SSM, including language modeling tasks, we confirm that appropriate initialization allows the recurrent parameters to remain fixed during training, leading to more stable training of the entire network and higher performance than the conventional training scheme. Our results partially elucidate the learning mechanisms of SSMs and are expected to contribute to the development of improved recurrent neural network models.

1 Introduction

State space models (SSMs), a class of recurrent neural networks (RNNs), have shown strong performance in time series modeling [1, 2]. Prior work reports that in language tasks, SSMs outperform Transformers [3], the current standard in language modeling, in several aspects, including performance and computational efficiency [1, 2]. These advances are considered to stem from the essential architecture of SSMs, which mitigates the training difficulties inherent in RNNs. However, some key components still lack a solid theoretical foundation explaining how they address conventional problems, such as the vanishing and exploding gradient problems [4–6] and a large number of model parameters.

Prior work on vanishing gradients focused on the decay of error signals through repeated gradient multiplication during backpropagation through time (BPTT). To illustrate this, we considered a simple case in a linear SSM with random inputs and computed the gradient with respect to the linear

[*]Correspondence to: kan@isi.imi.i.u-tokyo.ac.jp

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

readout weights under mean squared error (MSE):

$$G = \bar{B}\boldsymbol{V}^\top \bar{\boldsymbol{e}}, \tag{1}$$

where $\bar{B}$ is a constant coefficient, $\bar{\boldsymbol{e}} = (\bar{e}_{T-1} \quad \cdots \quad \bar{e}_0)^\top$ is the error signal passed from the upper layer, $\boldsymbol{V} = (\boldsymbol{V}_{T-1} \quad \cdots \quad \boldsymbol{V}_0)^\top$ is a Vandermonde matrix, and T is the sequence length (see Section 3.2). The vector $\boldsymbol{V}_\tau = (\lambda_1{}^\tau \quad \cdots \quad \lambda_N{}^\tau)^\top$ is determined by the eigenvalues $\{\lambda_i\}_{i=1}^N$ of the recurrent weight. Because for each temporal delay τ, the error term $\bar{e}_\tau$ is multiplied by the corresponding $\boldsymbol{V}_\tau$, previous studies have stated that the error propagation exhibits exponential decay, which negatively affects learning [7, 8]. This problem may be mitigated in two ways.

One approach is to adopt an initialization that alleviates the decay [9, 10]. In the context of SSMs, it is suggested that the weights should be initialized with the theoretically longest memory [11]. However, as the length of a network's memory increases, the output becomes more sensitive to small parameter changes, making training more difficult [4]. In addition, with regard to RNNs, an upper bound on memory capacity [12] exists. Therefore, achieving the longest memory requires sacrificing memory accuracy at each time step. While empirical results suggest that memory length should be prioritized [11, 13], a theoretical justification is still lacking.

Another possibility is that training itself could mitigate information loss. For example, $|\lambda_i|$ could approach 1 during training, reducing exponential decay. In this case, early-stage vanishing gradients owing to an inappropriate initialization would not be problematic. Though some works already suggest that overcoming vanishing gradients is generally difficult [7, 8], theoretical proof is still lacking.

Furthermore, as the learning mechanisms of SSMs are not fully understood, it remains unclear which components of SSMs are essential. In particular, to reduce computational cost, recurrent weights are often reduced to diagonal matrices [14, 2]. However, the necessity of eigenvectors has not been theoretically ruled out, and there is no proof establishing the equivalence between two different networks whose recurrent weights are nondiagonal and diagonal [13, 14] in terms of information processing [15].

Prior studies theoretically analyzing the learning dynamics of linear RNNs have also not solved these issues [16–18]. Addressing these questions could lead to better models and more effective training approaches, which may help further mitigate typical machine learning problems, including overfitting [19, 20] and a large number of training steps. To analyze the learning dynamics of SSMs, we evaluate the information processing characteristics of systems in response to input quantitatively and compare them across models using information processing capacity (IPC) [15]. This indicator statistically measures how a time-evolving information processing system transforms arbitrary inputs and retains the processed results within the state as memory, including both linear and nonlinear transformations. IPC can be used for any information processing system that processes inputs over time, regardless of the system's internal structure. When calculated with delayed linear inputs, IPC is called the memory function (MF) [12, 21–25]. We can comprehensively investigate the information processing of linear RNNs using MF, which serves as a fundamental and effective metric for evaluating information processing properties in linear models.

Contributions In this study, we clarify the learning mechanism of SSMs in three stages.

First, we evaluate the information processing of SSMs using MF and reformulate linear SSMs in terms of their essential components by proving that the input weight and the eigenvectors of the recurrent weight matrix can be eliminated (Section 3.1). This establishes a foundation for analyzing SSMs through the lens of MF.

Second, we demonstrate that focusing solely on error signals is insufficient to explain a loss of supervisory information (Section 3.2). We express the error $\bar{\boldsymbol{e}}$ (Eq. 1) as a function of the input history and express the gradients as functions of the input history, under the assumption that the model output is inherently a function of the input history and that the target is likewise a function of the input history. Under this formulation, we can capture the loss of supervisory information that is overlooked in conventional vanishing gradient analyses and show that the error $\bar{\boldsymbol{e}}$ is described as $\bar{\boldsymbol{e}} = (\hat{v} - \boldsymbol{M}\boldsymbol{\alpha})$, where $\hat{v} \in \mathbb{R}^T$ and $\boldsymbol{\alpha} \in \mathbb{R}^T$ correspond to the SSM output and supervisory signals, respectively. The diagonal entries of matrix $\boldsymbol{M} \in \mathbb{R}^{T\times T}$ correspond to the MF of the SSM, and its off-diagonal entries are suppressed by the MF. This result shows that even when $\bar{\boldsymbol{e}} \neq \boldsymbol{0}$, the supervisory information might not guide the parameter update direction because $\boldsymbol{M}\boldsymbol{\alpha} = \boldsymbol{0}$ may

hold, while generally, $\hat{v} \neq \mathbf{0}$. As $\boldsymbol{\alpha}$ is associated with past inputs, the term $\boldsymbol{M}\boldsymbol{\alpha}$ represents the inner product between the memory supplied by the SSM and the memory demanded by the supervisory signals. Therefore, the condition $\boldsymbol{M}\boldsymbol{\alpha} = \mathbf{0}$ indicates that there is no overlap between these memories. Thus, discarded supervisory components do not contribute to the readout gradient at the current parameters. The fact that $\boldsymbol{M}$ is determined only by the eigenvalues of $\boldsymbol{A}$ highlights the essential role of initialization in determining $\boldsymbol{A}$. Moreover, it is revealed that for successful learning, the initial $\boldsymbol{M}$ should prioritize memory length over memory accuracy; otherwise, crucial label information may be lost during backpropagation through one SSM layer. Combining this theoretical result and the fact that MF represents the information processing of SSMs, we propose that the recurrent weight may not require training if SSMs are initialized with weights that satisfy the memory requirements.

Finally, we empirically validate our theoretical results by evaluating performance across all tasks in the Long Range Arena (LRA) benchmark [26] and demonstrate that our proposed method is effective with linear SSMs [13, 14]. We also apply the results to a nonlinear SSM, Mamba [1], and show that the method is also effective in language modeling (Section 3.3 and Section 3.4). Compared with training the recurrent weights, the proposed method yields faster convergence, mitigates overfitting, and offers greater potential for accuracy improvements.

2 Background

State space models SSMs with N-dimensional states are formulated as follows [1]:

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}_k \boldsymbol{x}_k + \boldsymbol{B}_k \boldsymbol{u}_k \qquad \boldsymbol{y}_k = \tilde{\boldsymbol{C}}_k \boldsymbol{x}_k, \tag{2}$$

where $\boldsymbol{x}_k \in \mathbb{R}^N$, $\boldsymbol{u}_k \in \mathbb{R}$, $\boldsymbol{y}_k \in \mathbb{R}$, $\boldsymbol{A}_k \in \mathbb{R}^{N\times N}$, $\boldsymbol{B}_k \in \mathbb{R}^{N\times 1}$, and $\tilde{\boldsymbol{C}}_k \in \mathbb{R}^{1\times N}$ denote the sampled state, input, output, recurrent weight, input weight, and output weight at discrete time k, respectively. In linear SSMs, the weights are time invariant and do not depend on the time step k. Linear SSMs designed to efficiently capture long-range dependencies with deep architectures are referred to as structured state space sequence models (S4) [13]. Their recurrent weights are initialized according to the high-order polynomial projection operators (HiPPO) [11] framework, which achieves the theoretically longest memory under the Lebesgue measure. While S4 allows nonzero off-diagonal entries in the recurrent matrix $\boldsymbol{A}$, S4D [14] restricts $\boldsymbol{A}$ to be diagonal. In contrast, Mamba [1] employs input-dependent weights and can be viewed as a nonlinear RNN; thus, it is treated as a nonlinear SSM in this work.

Memory capacity We evaluate the information processing in the discretized SSM using MF [12, 15], which represents how well the delayed input series $u_{t-\tau}$ can be reconstructed from the current network state. Any independent and identically distributed (IID) input can be employed when evaluating the MF. Following previous studies, we sample each input u_t independently from a uniform distribution on $[-1, 1]$ in numerical experiments. The delayed input is reconstructed using the linear readout $\hat{u}_{t-\tau} = \boldsymbol{w}_{\text{out}}^\top \boldsymbol{x}_t$, where τ is the delay from the current time. The readout weight vector $\boldsymbol{w}_{\text{out}}$ is determined by minimizing MSE: $L_{\text{MSE}} = \langle (\hat{u}_{t-\tau} - u_{t-\tau})^2 \rangle$, where $\langle \cdot \rangle = \frac{1}{T}\sum_{t=1}^{T}(\cdot)$ denotes the time average and T is the number of sampled time steps. Accordingly, the MF is defined as

$$M[u_{t-\tau}] = 1 - \frac{\min_{\boldsymbol{w}_{\text{out}}} \langle (\hat{u}_{t-\tau} - u_{t-\tau})^2 \rangle}{\langle u_{t-\tau}^2 \rangle} \ (\leq 1). \tag{3}$$

The upper bound 1 is attained when the system fully memorizes the target, i.e., the delayed input series. The sum of the MF with respect to all τ represents the memory capacity (MC) of the SSM:

$$M_{\text{sum},u} = \sum_{\tau=0}^{\infty} M[u_{t-\tau}] \ (\leq N). \tag{4}$$

The upper bound of $M_{\text{sum},u}$ is determined by the number of linearly independent time series in the state, which is called the rank and is ideally N. This implies that there is a trade-off between the accuracy and length of memory. By calculating Eq. 3, Dambre et al. [15] derived an alternative expression for the MF: $M[u_{t-\tau}] = \boldsymbol{U}_\tau^\top \boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{U}_\tau / (\boldsymbol{U}_\tau^\top \boldsymbol{U}_\tau)$, where $\boldsymbol{X} \in \mathbb{R}^{T\times N}$ is a matrix whose columns represent the state time series, and $\boldsymbol{U}_\tau \in \mathbb{R}^T$ is the delayed input series. The MF and MC of linear RNNs were analyzed in a previous study [21]:

$$M[u_{t-\tau}] = \boldsymbol{V}_\tau{}^\top (\boldsymbol{V}^\top \boldsymbol{V})^{-1} \boldsymbol{V}_\tau \qquad M_{\text{sum},u} = \text{tr}\left[\boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top\right], \tag{5}$$

where $\boldsymbol{V}$ is a Vandermonde matrix (Eq. 1) determined by the eigenvalues of $\boldsymbol{A}$ in Eq. 2. The maximum eigenvalue magnitude is called the spectral radius $\rho(\boldsymbol{A})$, which is a crucial parameter for the dynamics of linear RNNs. If this value is greater than 1, the states will diverge as the time step progresses.

Gradient-based learning One of the most widely used parameter update strategies in machine learning is based on the principle of gradient descent. Variants such as stochastic gradient descent (SGD) [27, 28] and Adam [29] exist. In all of these methods, errors propagate from the final layer backward, and the gradients computed at each layer determine the direction of parameter updates. Here, the error at the final layer is defined as $L_o(\hat{\boldsymbol{y}}_o, \boldsymbol{y}_o)$, where $\hat{\boldsymbol{y}}_o \in \mathbb{R}^{H_o}$ is the model output, $\boldsymbol{y}_o \in \mathbb{R}^{H_o}$ is the target label, and H_o is the dimensionality of the desired output. During training, to evaluate the error of the previous layer and determine the parameter update direction, the gradient $\nabla_{\boldsymbol{\theta}_o} L_o(\hat{\boldsymbol{y}}_o, \boldsymbol{y}_o)$ of the final layer is calculated, where $\boldsymbol{\theta}_o$ denotes the parameters of the output layer. Subsequently, the gradient for the layer immediately preceding the output layer is computed, a process that is applied recursively across layers. Accordingly, the optimal output of the lth layer is determined by the gradient passed from the $(l+1)$th layer, suggesting that we can define the loss function $L_l(\hat{\boldsymbol{y}}_l, \boldsymbol{y}_l)$ for the lth layer in a manner similar to the final layer, where $\hat{\boldsymbol{y}}_l$ and $\boldsymbol{y}_l$ represent the actual and ideal outputs of that layer, respectively.

In this study, we assume that the loss function L_l of the SSM layer can be represented by the MSE. Accordingly, the gradient of the SSM is described by

$$\nabla_{\boldsymbol{\theta}_l} L_l(\hat{\boldsymbol{y}}_l, \boldsymbol{y}_l) = \nabla_{\boldsymbol{\theta}_l} \frac{1}{H_l} ||\hat{\boldsymbol{y}}_l - \boldsymbol{y}_l||^2, \tag{6}$$

where $\boldsymbol{\theta}_l$ denotes the parameters of the lth layer, and H_l is the dimensionality of the layer output.

3 Results

3.1 MF of SSMs

In this section, we show why MF is an effective indicator for SSMs in two steps. First, we analytically simplify the linear SSM based on the theoretical results of MF obtained in previous studies. Second, we numerically demonstrate the MF of SSM architectures proposed by prior research and explain their memory profiles in terms of MF. For the analytical results, assume that $\boldsymbol{A} = \boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^{-1}$ is diagonalizable, $\rho(\boldsymbol{A}) < 1$, every entry of $\boldsymbol{P}^{-1}\boldsymbol{B}$ is nonzero, and $\boldsymbol{V}$ has full column rank.

S4D is theoretically equivalent to S4 According to a previous study [15], all information processing of linear SSMs can be comprehensively measured by MF. The state $\boldsymbol{x}_t$ of linear RNNs can be described by $\boldsymbol{x}_t = \sum_{\tau=0}^{\infty} \boldsymbol{\chi}_\tau u_{t-\tau}$, where $\{\boldsymbol{\chi}_\tau\}_{\tau=0}^{\infty}$ is a sequence of fixed vectors as in Eq. 23. This indicates that the output $\hat{y}_t$ from the readout is represented by $\hat{y}_t = \sum_{\tau=0}^{\infty} \chi_\tau u_{t-\tau}$ using a sequence of fixed coefficients $\{\chi_\tau\}_{\tau=0}^{\infty}$, suggesting that linear RNNs perform information processing solely by computing a linear combination of past inputs. Since the MF $M[u_{t-\tau}]$ is proven to be analytically equivalent to the norm of $\boldsymbol{\chi}_\tau$ [30], it can fully capture the information processing characteristics of linear SSMs. Combining this result with the analytical solution derived by Guan et al. [21] (Eq. 5) reveals that the MF depends only on the eigenvalues of $\boldsymbol{A}$ (Eq. 2). This proves that both the input weight $\boldsymbol{B}$ and the eigenvectors of $\boldsymbol{A}$ can be freely specified, demonstrating that S4 is theoretically equivalent to S4D because they share the same eigenvalues. Accordingly, we redefine the recurrent equation of the discrete SSM:

$$\boldsymbol{x}_{k+1} = \boldsymbol{\Lambda}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k, \tag{7}$$

where $\boldsymbol{\Lambda} = \mathrm{diag}\begin{pmatrix}\lambda_1 & \lambda_2 & \cdots & \lambda_N\end{pmatrix}$. Unless otherwise specified, we use an input weight $\boldsymbol{B} = B\boldsymbol{1}$, where $\boldsymbol{1} \in \mathbb{R}^{N\times 1}$ is the vector whose elements are all 1 and B is an arbitrary nonzero value.

MF of HiPPO eigenvalues To demonstrate the effectiveness of MF, we calculated MF for five typical eigenvalue realizations and analyzed the corresponding memory structures. Two groups of eigenvalues for the SSMs were employed. The first group consisted of those proposed in previous studies [14] to realize the longest memory structures—specifically, "S4Dinv" and "S4Dlin." The second group consisted of other representative eigenvalues: "random," "linear," and "step." In this study, "random" refers to eigenvalues of matrix $\boldsymbol{A}$ randomly generated from a uniform distribution, while "linear" indicates real eigenvalues that are evenly spaced, and "step" indicates that all eigenvalues are

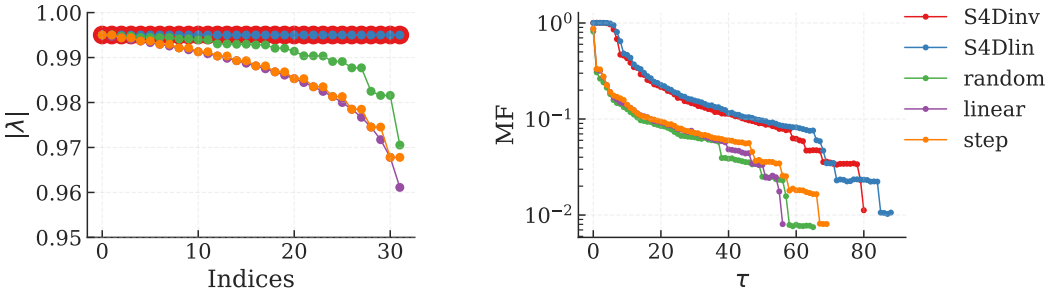


Figure 1: The MF of five eigenvalue realizations. The left (right) panel shows the eigenvalue magnitudes (MF). The horizontal axis is the index of eigenvalues (delay τ of input). The system size N is 32, and the results were obtained by calculating Eq. 3, where $T = 1024$, following what was typically used in previous studies' models.

complex conjugate pairs whose magnitudes are arranged in a linear distribution. We evaluated the MFs based on the actual setting used for training and evaluating the model. In principle, one SSM layer consists of multiple distinct SSMs. To summarize the individual MFs in the lth layer, we define the supremum MF as follows:

$$M[u_{t-\tau}]^{(\ell)} = \sup M[u_{t-\tau}]_i^{(\ell)} = \max_i M[u_{t-\tau}]_i^{(\ell)}, \tag{8}$$

where i is the index identifying a certain SSM. Fig. 1 shows the eigenvalues and the MF of one layer (Eq. 8) and illustrates that "S4Dinv" and "S4Dlin" exhibited longer and stronger memory than other eigenvalue realizations, suggesting that they possess richer information processing. Clearly, the sums of MFs differ. This can be explained by the fact that the MF sum does not numerically reach its upper bound, namely the number of nodes N, and that the system rank varies depending on the eigenvalues. Therefore, longer MFs are attributed to the characteristics of eigenvalues; "S4Dinv" and "S4Dlin" inherently generate higher system ranks. From the perspective of the MF, this eigenvalue group can be considered superior to the other group. We provide the motivation for choosing these configurations and the detailed procedure of analysis in Appendices B and C.

3.2 Importance of initial parameters for successful learning

Next, we theoretically analyze the learning process by focusing on the parameter update strategy of linear SSMs. To simplify the problem, we assume that the input u_t is randomly sampled from a uniform distribution, consistent with the setting used for the MF. The analytical gradient formulas below additionally use the idealized input-orthogonality condition stated in Appendix D. Given a target sequence $\boldsymbol{y} = \{y_i\}_{i=1}^T$, the loss function of a single SSM layer is described by $L_{\text{MSE}} = \frac{1}{T}\sum_{i=1}^T (f(\boldsymbol{\theta}^\top \boldsymbol{x}_i) - y_i)^2$, where T is the sequence length, $\boldsymbol{\theta}$ is the parameter vector of the linear readout layer, $f(\cdot)$ is the activation function applied elementwise after the linear readout, and $\boldsymbol{x}_i$ is the state vector at time i. The gradient $G(\boldsymbol{y})$ with respect to the readout weights is given by

$$G(\boldsymbol{y}) = \nabla_\theta L_{\text{MSE}} = \frac{2}{T}\boldsymbol{X}^\top \left[(f(\boldsymbol{X\theta}) - \boldsymbol{y}) \odot f'(\boldsymbol{X\theta})\right]. \tag{9}$$

To gain intuition about the factors governing parameter updates, we first focus on the SSM before the nonlinear readout is applied. In the case in which f is linear, the gradient is described as follows:

$$G(\boldsymbol{y}) = \frac{2}{T}\boldsymbol{X}^\top(\boldsymbol{X\theta} - \boldsymbol{y}). \tag{10}$$

Following the approach of the MF, we consider a task to predict a delayed input series $\boldsymbol{y} = \boldsymbol{U}_\tau \in \mathbb{R}^T$. The gradient in this task is computed as $G(\boldsymbol{U}_\tau) = 2\langle u^2\rangle B\boldsymbol{V}^\top(B\boldsymbol{V\theta} - \boldsymbol{V}(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}_\tau)$. Because a linear SSM computes only linear combinations of delayed inputs, its possible outputs can be written as $\hat{\boldsymbol{y}} = \sum_{k=0}^{T-1}\chi_k\boldsymbol{U}_k$, where $\{\chi_\tau\}_{\tau=0}^{T-1}$ is a series characterizing $\hat{\boldsymbol{y}}$. Therefore, under any learning procedure, the optimized output $\hat{\boldsymbol{y}} = \sum_{k=0}^{T-1}\alpha_k\boldsymbol{U}_k$ can also be expressed by a sequence $\{\alpha_\tau\}_{\tau=0}^{T-1}$. Here, we regard $\sum_{k=0}^{T-1}\alpha_k\boldsymbol{U}_k$ as the target sequence $\boldsymbol{y} = \sum_{k=0}^{T-1}\alpha_k\boldsymbol{U}_k$, which describes all possible

target sequences for linear SSMs. Substituting this expression for the target sequence $\boldsymbol{y}$ into the readout gradient gives

$$G(\boldsymbol{y}) = 2\langle u^2 \rangle B\boldsymbol{V}^\top (B\boldsymbol{V}\boldsymbol{\theta} - \boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top \boldsymbol{\alpha}). \tag{11}$$

In Section 1, $B\boldsymbol{V}\boldsymbol{\theta}$ and $\boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top$ are denoted by $\hat{v}$ and $\boldsymbol{M}$, respectively. The derivation of these formulas is shown in Appendix D. Eq. 11 shows that the state series $\boldsymbol{X}$ (Eq. 10) is replaced by the matrix $B\boldsymbol{V}$, where $\boldsymbol{V}$ is the Vandermonde matrix defined in Eq. 5. The matrix $\boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top$, which is the normalized version of $\boldsymbol{V}^\top$, acts as a weighting function for $\boldsymbol{\alpha}$. The τth element of $\boldsymbol{\alpha}$ multiplies the τth column of $\boldsymbol{V}^\top$, denoted as the vector $\boldsymbol{V}_\tau$, which represents the decay of input information from τ steps ago. Because MF measures the decay of input information stored in the state time series and $\boldsymbol{V}$ represents $\boldsymbol{X}$, the decay represented by $\boldsymbol{V}$ can be evaluated through the MF. In fact, the diagonal components of the matrix $\boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top$ are the MF. This result offers two insights. First, the MF describes not only the decay of input information in the forward pass but also the decay of error information in the backward pass, as given by $B\boldsymbol{V}^\top$ (see Appendix F.1). Second, when the target sequence $\boldsymbol{y}$ contains past input information α_k, but the corresponding memory $M[u_{t-k}]$ is not contained in the SSM, the supervisory information α_k is multiplied by 0, meaning that such information is not reflected in the readout gradient.

Memory requirement that must be realized at initialization Consider a case in which $\boldsymbol{y}$ is solely composed of inputs from the distant past for which the MF is approximately zero at the corresponding delays. Because

$$\boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top \boldsymbol{\alpha} \approx \boldsymbol{0} \tag{12}$$

holds in this case, the target $\boldsymbol{y}$ does not contribute significantly to updates of the readout weights, which also appear in the gradient with respect to Λ. This indicates that the model has a limited ability to update parameters (Theorem D.4), and motivates choosing an initialization with sufficiently long memory for the task. A nonzero supervisory contribution from the target $y_t = u_{t-k}$ requires nonzero MF at delay k (Lemma D.5). This suggests that the model should preserve memory at task-relevant delays, potentially at the expense of accuracy at other delays (Theorem D.6). To learn tasks that rely on information from arbitrarily distant past inputs, an SSM should be initialized with an infinitely long memory (Lemma D.5). According to the trade-off between the accuracy and length of memory (Eq. 4), we must satisfy this condition by sacrificing some memory accuracy. Our results suggest that this condition may be given higher priority than memory accuracy because gradient updates are still executed using supervisory signals, even when the memory is very weak (Theorem D.6). However, if accuracy is prioritized, supervisory signals from the distant past are likely to be discarded, and effective updates may not progress. The meaning of the product $\boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top \boldsymbol{\alpha}$ is further analyzed in Appendix D.3.

SSM with nonlinear output layer We extend the analysis to the gradient of a nonlinear readout because the input to the readout layer depends entirely on the output of the SSM. To illustrate how nonlinearity affects the output of the full SSM layer, we provide the gradient formula for a nonlinear readout here (Theorem D.1). In the case where f is a nonlinear activation function, the gradient (Eq. 9) is described by

$$G(\boldsymbol{y}) \quad = \quad \frac{2}{T}\boldsymbol{X}^\top \left[f(\boldsymbol{X}\boldsymbol{\theta}) \odot f'(\boldsymbol{X}\boldsymbol{\theta}) - \hat{\boldsymbol{U}}^\top \boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top \bar{\boldsymbol{\alpha}} \right], \tag{13}$$

where $\hat{\boldsymbol{U}} = (\boldsymbol{U}_{T-1} \quad \cdots \quad \boldsymbol{U}_0)$. Let $\bar{\boldsymbol{y}} = \boldsymbol{y} \odot f'(\boldsymbol{X}\boldsymbol{\theta})$ denote the target sequence modified by the activation derivative. Assuming that the target sequence $\boldsymbol{y}$ is a function of the input history, we use the coefficient vector $\bar{\boldsymbol{\alpha}}$ to characterize this modified target sequence. We can confirm that the update direction is determined by the vector $\boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top \bar{\boldsymbol{\alpha}}$. Therefore, we can elucidate the update direction by the product of MF and the modified label series similarly. The result also applies to a nonlinear readout. Appendix D.5 discusses a related composite loss.

Training eigenvalues For linear RNNs, BPTT explicitly shows that the error signals driving recurrent-weight updates depend on the current readout weights (Appendix F.1). Thus, learning the recurrent weights, including their eigenvalues, is coupled to readout learning. Here, we consider the SSM layer that learns the eigenvalues of the internal weight matrix. In this case, the MF changes during the training process, and the memory within certain time regions may be lost because the sum of the MF is theoretically constant (Eq. 4). To increase the MF within a specific time region, it is analytically necessary to reduce the MF in other regions. In practice, we cannot guarantee that the lost

memory will not be needed later in training. For example, when we use the structured eigenvalues (i.e., "S4Dinv" and "S4Dlin" [14]), their theoretically longest memory structure can be destroyed. However, the advantages of learning eigenvalues can also be explained using MF in terms of both shortening and extending memory. Shortening the MF is beneficial in tasks in which only short memory is required (Hypothesis D.7). It is preferable to concentrate on essential memory rather than sustain low-accuracy memory over long delays, because the sum of the MF is theoretically fixed. The benefits of extending the MF have already been discussed above. As eigenvalues are adapted to alter the organization of existing memory, an unintended extension of the MF (Definition D.8) may also occur (Hypothesis D.9). For example, enhancing the MF at $\tau = 80$ during learning with the eigenvalues of "S4Dinv" in Fig. 1 can induce a nonzero MF at $\tau = 100$, which is initially 0, once a threshold is exceeded (see Appendix C). However, this is not guaranteed. To determine which learning strategy is more effective, numerical experiments are conducted in the next section.

3.3 SSM with fixed eigenvalues

To address the question raised in the previous section, we propose using an SSM with fixed eigenvalues and training only its output weights. We refer to this setup as the reservoir computing (RC) setting, following the RC approach in which recurrent and input weights remain fixed and only output weights are learned [31–35]. This approach preserves the memory structure, preventing the loss of memory due to learning. To demonstrate the effectiveness of this learning method and to investigate how initializing with the longest memory structure affects learning, we evaluate all five eigenvalue initializations for the SSM. As benchmark tasks, we employed the LRA benchmark, which has been widely used in prior work [13, 14] and requires long-term memory to solve. For all tasks and under each eigenvalue realization, we computed task accuracies in two settings: (1) a setting in which eigenvalues were allowed to be learned and (2) the RC setting. To examine the robustness of each model, we trained it using six different random seeds. Table 1 presents the accuracies obtained under each condition. Except for the random seed, all model hyperparameters were kept identical to those used in previous work [13]. Detailed values are provided in Appendix E.

Table 1: Test accuracies on LRA tasks for S4D with trainable or fixed eigenvalues under five eigenvalue initializations, with S4 included as a reference. Each model was evaluated on each task using six different random seeds to investigate whether the models were robust. The minimum, maximum, mean, and standard deviation across runs are reported. For reference, we also report the S4 results under the same setting. Three settings are presented: "S4," "learnable eigenvalues," and "RC." Accuracies are underlined when a given eigenvalue realization achieves the highest accuracy across each indicator. In addition, when "RC" surpasses "learnable eigenvalues," the corresponding accuracy is bolded. The exact model architecture is described in Algorithm 1.

Eigenvalues	ListOps max acc	ListOps min acc	ListOps ave (std)	Text max acc	Text min acc	Text ave (std)	Retrieval max acc	Retrieval min acc	Retrieval ave (std)	Image max acc	Image min acc	Image ave (std)	Pathfinder max acc	Pathfinder min acc	Pathfinder ave (std)	PathX max acc	PathX min acc	PathX ave (std)
S4	0.602	0.578	0.585 (0.008)	0.872	0.866	0.869 (0.002)	0.910	0.907	0.908 (0.001)	0.874	0.871	0.872 (0.001)	0.934	0.919	0.928 (0.006)	0.960	0.937	0.951 (0.008)
S4Dinv	0.594	0.589	0.592 (0.002)	0.853	0.839	0.845 (0.005)	0.863	0.760	0.812 (0.037)	0.864	0.859	0.861 (0.002)	0.910	0.877	0.897 (0.013)	0.922	0.903	0.909 (0.007)
S4Dlin	0.598	0.587	0.591 (0.004)	0.848	0.830	0.839 (0.006)	0.843	0.750	0.803 (0.031)	0.868	0.860	0.866 (0.003)	0.896	0.865	0.886 (0.011)	0.926	0.912	0.918 (0.006)
random	0.552	0.476	0.531 (0.026)	0.857	0.835	0.845 (0.008)	0.903	0.899	0.902 (0.002)	0.820	0.815	0.818 (0.002)	0.870	0.846	0.860 (0.007)	0.877	0.841	0.855 (0.012)
linear	0.543	0.488	0.512 (0.020)	0.839	0.811	0.829 (0.009)	0.899	0.894	0.897 (0.002)	0.782	0.768	0.773 (0.004)	0.797	0.754	0.780 (0.014)	0.820	0.503	0.684 (0.119)
step	0.605	0.575	0.586 (0.010)	0.862	0.852	0.858 (0.003)	0.883	0.841	0.863 (0.015)	0.789	0.779	0.782 (0.003)	0.858	0.824	0.844 (0.013)	0.840	0.824	0.832 (0.005)
RC S4Dinv	0.585	0.570	0.576 (0.005)	**0.857**	**0.846**	**0.853 (0.004)**	**0.906**	**0.901**	**0.904 (0.002)**	**0.873**	**0.865**	**0.868 (0.003)**	0.899	**0.889**	0.893 (0.004)	**0.932**	**0.923**	**0.926 (0.003)**
RC S4Dlin	0.593	0.577	0.582 (0.006)	**0.850**	**0.843**	**0.845 (0.003)**	**0.903**	**0.898**	**0.902 (0.002)**	**0.877**	**0.871**	**0.873 (0.002)**	0.892	**0.872**	0.884 (0.008)	**0.933**	**0.922**	**0.929 (0.004)**
RC random	0.535	**0.477**	0.510 (0.019)	0.845	0.813	0.834 (0.010)	0.901	0.896	0.899 (0.002)	0.733	0.698	0.713 (0.012)	0.824	0.777	0.802 (0.016)	0.820	0.773	0.797 (0.016)
RC linear	0.530	0.423	0.495 (0.035)	0.815	0.801	0.807 (0.005)	0.894	0.889	0.892 (0.002)	0.686	0.634	0.665 (0.017)	0.769	0.506	0.668 (0.115)	0.669	**0.504**	0.567 (0.057)
RC step	0.556	0.500	0.530 (0.021)	0.845	0.833	0.838 (0.004)	**0.903**	**0.897**	**0.900 (0.002)**	0.745	0.740	0.742 (0.001)	0.832	0.793	0.810 (0.013)	0.813	0.504	0.739 (0.107)

Our experiments show that the performance in the RC setting does not degrade substantially across all of the LRA tasks when the eigenvalues are initialized with structured memory. In addition, fixing

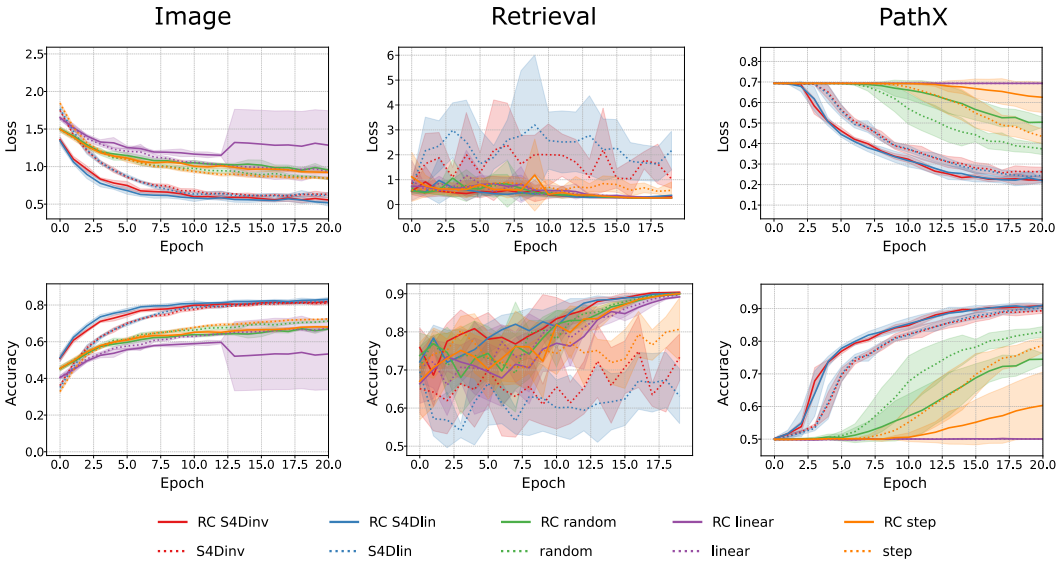


Figure 2: Test loss and accuracy of three tasks. The top (bottom) row indicates test loss (accuracy), and each column corresponds to a different task. The horizontal axes represent epochs. Lines represent averages across seeds, with shaded regions indicating standard deviations. Solid lines represent models in the RC setting, while dotted lines indicate models with trainable eigenvalues. Each color represents a different eigenvalue realization.

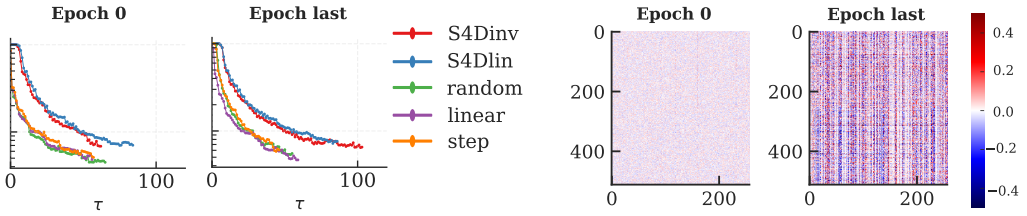


Figure 3: The supremum MF (left panels) and the linear readout weights $\boldsymbol{W}_{out}$ (right panels) at the last layer before and after training. In the left panels, the horizontal (vertical) axis shows the input delay τ (MF). The colors indicate the eigenvalue realizations. The state size N is 32, and the results are obtained by calculating Eq. 3 with $T = 1024$. In the right panels, the horizontal (vertical) axis represents the indices of SSMs from the following (preceding) layer. The preceding layer has twice as many indices as the following layer, owing to the bidirectional structure. As an example, we show the results for the Pathfinder task. The results of all layers are presented in Fig. 9 and Fig. 11 in the appendix.

the internal recurrent weights reduces the number of trainable parameters, which is shown in Table 3 in the Appendix. These results suggest that the RC setting is effective. Notably, in five tasks, the RC setting with structured eigenvalues yielded higher minimum accuracies than the setting with learnable eigenvalues. In contrast, the model with representative eigenvalues showed poorer performance in the RC setting, suggesting the importance of the initialization with structured eigenvalues. To examine the effects of the RC setting, we monitored how the loss and accuracy evolved over time. For models initialized with structured eigenvalues, the test loss for Image, Retrieval, and PathX converged to lower values more rapidly in the RC setting, and the test accuracy for Retrieval and PathX also increased notably faster (Fig. 2). In addition, for the Text, Retrieval, and Image tasks, the RC setting yielded test accuracy comparable to—or even higher than—that obtained with trainable eigenvalues. Moreover, the training accuracy showed a relatively modest enhancement in the RC setting, indicating better mitigation of overfitting (see Appendix E.1). For the remaining tasks, these trends were less pronounced, but no significant contradictory outcomes were observed.

To examine whether eigenvalue updates are necessary, we compared the MF and readout weights of SSMs before and after training when eigenvalue training was enabled. As shown in the left panels of Fig. 3, even after training, the MFs of the "random," "linear," and "step" configurations did not reach those of the structured eigenvalues before training. This indicates that a good initialization matters more than training the eigenvalues. Training did extend the MF slightly in some cases, and in ListOps this small gain seems to have outweighed the benefit of fixed eigenvalues, which would explain why the RC setting underperformed there. In all other tasks, however, the RC setting achieved higher minimum accuracy across seeds than the trainable-eigenvalue setting for both S4Dinv and S4Dlin (Table 1). Thus, when the initial eigenvalues already provide a long MF, fixing them during training is at least as effective as learning them, and often more effective. The right panels of Fig. 3 show how the readout weights change when eigenvalue training is enabled. They change substantially over the course of training, indicating that readout learning remains important even when the eigenvalues are trained. Together with the high accuracy of the RC setting, this suggests that learning in SSMs proceeds mainly through parameters other than the eigenvalues.

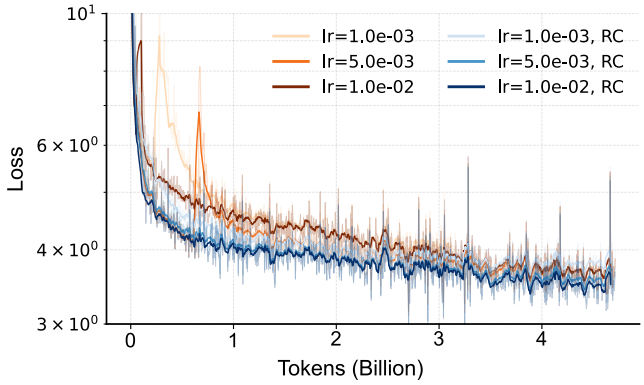


Figure 4: Training loss of Mamba on the language modeling task under the RC setting and full training setting at different learning rates. The horizontal axis shows the number of training tokens in billions. The exact model architecture is described in Algorithm 2.

3.4 Nonlinear SSM

Our numerical experiments demonstrate the effectiveness of the RC setting in Mamba, a nonlinear SSM. To examine this, we evaluate a Mamba-based architecture with frozen recurrent weights (RC setting), analogous to the setting used for linear SSMs. As shown in Fig. 4, the RC model not only learns successfully but also converges to a lower loss than the model with trainable recurrent weights. Examining the dependence on the learning rate, we observe that larger learning rates lead to earlier loss spikes, slower convergence, and higher final loss. This can be attributed to both the difficulty of effectively optimizing the memory structure and the instability of learning the recurrent structure, as in linear SSMs. Indeed, measurements of Mamba's MF show that its memory is longest at initialization (Appendix E.2). Moreover, memory accuracy, as measured by the magnitude of MF, shows little improvement over its initial value across all layers during training. The RC setting works well in Mamba possibly because it already has a sufficiently long MF at initialization, similar to "S4Dinv" and "S4Dlin" in S4D.

4 Discussion and Conclusion

In this study, we analyzed the learning dynamics of SSMs using MF by focusing on gradient computation. Previous studies imposed constraints on $\boldsymbol{A}$ without sufficient theoretical justification. For example, the original S4D results [14] apply only when $\boldsymbol{A}$ are HiPPO matrices, while other studies treat $\boldsymbol{A}$ as random [18] and restrict or ignore its eigenvectors [16, 17]. Restricting eigenvectors limits applicability to specific models, and ignoring them confines the results to S4D, whose results cannot be applied to S4. In contrast, our analysis requires no specific eigenvector structure for $\boldsymbol{A}$ beyond the assumptions in Section 3.1, thereby extending the applicability of our results to a wider range of models. We provide a theoretical justification for disregarding the eigenvectors of $\boldsymbol{A}$, allowing us to

focus solely on the eigenvalues. Furthermore, because our results rely only on gradients, they are not limited to any specific training method and can be applied to any gradient-based optimization approach. The broader applicability of our results is discussed in Appendix F.3.

Building on this theoretical foundation, we focused on identifying the factors that determine whether gradient-based optimization methods can successfully learn. To this end, we considered how inputs affect the gradient. As a result, it was revealed that the MF, which describes the degree of decay over a delay of τ, weights the supervisory information corresponding to the τ-delayed input. Therefore, the MF acts as a filter on the supervisory signal, revealing how supervisory information can be lost. This naturally leads to the question of whether backpropagation helps in recovering from such a failure. Our results show that backpropagation exhibits only limited effectiveness in addressing this issue (Section 3.2 and Fig. 3). Further discussions are provided in Appendix F.1.

Given that the MF is explicitly incorporated in gradient computation, it becomes possible to design models using the MF as an indicator of their memory structure. However, there is currently no established method for designing a desired MF, and the constraints on the shapes of the MF remain unclear, which leaves two open problems for future research. First, our experiments did not isolate the benefit of memory length from that of memory accuracy, because extending memory also improved its accuracy under the linear SSM formulation. The trade-off between these properties remains task dependent and requires further investigation. These indicate that while a long and low-accuracy MF is currently a theoretical requirement, it is also practically feasible. Therefore, the claim discussed here remains a theoretical prediction. The applicability of the MF is further discussed in Appendix C.3. Second, it is unclear whether structured eigenvalues based on HiPPO (e.g., "S4Dinv" and "S4Dlin") are indeed the best choice from the perspective of MF just because HiPPO matrices guarantee the theoretically longest memory according to a different measure. As more favorable MFs are obtained when eigenvalue learning is allowed, developing algorithms that explicitly optimize MF may enable the discovery of models with improved memory properties.

To validate the learning mechanism uncovered in this study, we propose the RC setting, which fixes the eigenvalues, and show its effectiveness for both linear and nonlinear SSMs. We found that in the LRA tasks, up to approximately 14% of the total parameters were removed from the set of learnable parameters (see Appendix E). Stabilized training and improved performance were observed in both LRA and language modeling tasks. The RC setting can also be recommended from the perspective of preferable eigenvalues. When eigenvalues are designed to realize long memory structures, such as those derived from HiPPO, the magnitudes of these eigenvalues tend to approach 1. Because values exceeding 1 lead to output explosion, learning can make only slight adjustments to these eigenvalues, which may support the recommendation of the RC setting (see Appendix E.1). In other words, the system appears to be highly sensitive and potentially prone to structural collapse, which may explain the stabilization of S4D and Mamba under the RC setting. However, one scenario in which the RC setting may not be advisable is when solving tasks that require only extremely short-term memory. The reason for this was discussed at the end of Section 3.2, and additional analyses should take this into account. We believe that this study provides a theoretical foundation to elucidate the learning mechanisms of SSMs and will contribute to the development of improved RNN-based models.

Acknowledgments and Disclosure of Funding

This work was supported by a project (Grant Number JPNP14004) commissioned by the New Energy and Industrial Technology Development Organization (NEDO), by JST Moonshot R&D Program Japan Grant Number JPMJMS263O, by JST NEXUS Japan Grant Number JPMJNX26C5, and by JSPS KAKENHI Grant Number 25H01134. The authors declare no competing interests.

References

[1] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[2] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon,

U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[4] Nicolas Zucchet and Antonio Orvieto. Recurrent neural networks: vanishing and exploding gradients are not the end of the story. In *Advances in Neural Information Processing Systems*, volume 37, pages 139402–139443, 2024.

[5] Il Memming Park, Ábel Ságodi, and Piotr Aleksander Sokół. Persistent learning signals and working memory without continuous attractors. *arXiv preprint arXiv:2308.12585*, 2023.

[6] Piotr A Sokół, Ian Jordan, Eben Kadile, and Il Memming Park. Adjoint dynamics of stable limit cycle neural networks. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 884–887. IEEE, 2019.

[7] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[8] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr, 2013.

[9] Eugene Vorontsov, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal. On orthogonality and learning recurrent networks with long term dependencies. In *International Conference on Machine Learning*, pages 3570–3578. PMLR, 2017.

[10] Mikael Henaff, Arthur Szlam, and Yann LeCun. Recurrent orthogonal networks and long-memory tasks. In *International Conference on Machine Learning*, pages 2034–2042. PMLR, 2016.

[11] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1474–1487. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/102f0bb6efb3a6128a3c750dd16729be-Paper.pdf.

[12] H. Jaeger. *Short term memory in echo state networks*. GMD Forschungszentrum Informationstechnik, 2001. doi: 10.24406/publica-fhg-291107. URL https://publica.fraunhofer.de/handle/publica/291107.

[13] Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022.

[14] Albert Gu, Karan Goel, Ankit Gupta, and Christopher Ré. On the parameterization and initialization of diagonal state space models. *Advances in Neural Information Processing Systems*, 35:35971–35983, 2022.

[15] Joni Dambre, David Verstraeten, Benjamin Schrauwen, and Serge Massar. Information processing capacity of dynamical systems. *Scientific reports*, 2(1):514, 2012.

[16] Jakub Smékal, Jimmy TH Smith, Michael Kleinman, Dan Biderman, and Scott W Linderman. Towards a theory of learning dynamics in deep state space models. *arXiv preprint arXiv:2407.07279*, 2024.

[17] Alexandra Maria Proca, Clémentine Carla Juliette Dominé, Murray Shanahan, and Pedro AM Mediano. Learning dynamics in linear recurrent neural networks. In *Forty-second International Conference on Machine Learning*, 2024.

[18] Friedrich Schuessler, Francesca Mastrogiuseppe, Alexis Dubreuil, Srdjan Ostojic, and Omri Barak. The interplay between randomness and structure during learning in rnns. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 13352–13362. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/9ac1382fd8fc4b631594aa135d16ad75-Paper.pdf.

[19] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[20] Shaeke Salman and Xiuwen Liu. Overfitting mechanism and avoidance in deep neural networks. *arXiv preprint arXiv:1901.06566*, 2019.

[21] JingChuan Guan, Tomoyuki Kubota, Yasuo Kuniyoshi, and Kohei Nakajima. How noise affects memory in linear recurrent networks. *Physical Review Research*, 7(2):023049, 2025.

[22] Ali Rodan and Peter Tino. Minimum complexity echo state network. *IEEE transactions on neural networks*, 22(1):131–144, 2010.

[23] Igor Farkaš, Radomír Bosák, and Peter Gergel'. Computational analysis of memory capacity in echo state networks. *Neural Networks*, 83:109–120, 2016.

[24] Lukas Gonon, Lyudmila Grigoryeva, and Juan-Pablo Ortega. Memory and forecasting capacities of nonlinear recurrent networks. *Physica D: Nonlinear Phenomena*, 414:132721, 2020.

[25] Giovanni Ballarin, Lyudmila Grigoryeva, and Juan-Pablo Ortega. Memory of recurrent networks: Do we compute it right? *Journal of Machine Learning Research*, 25(243):1–38, 2024.

[26] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

[27] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[28] Léon Bottou et al. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, 91(8):12, 1991.

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] Tomoyuki Kubota, Hirokazu Takahashi, and Kohei Nakajima. Unifying framework for information processing in stochastically driven dynamical systems. *Physical Review Research*, 3(4): 043135, 2021.

[31] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2001.

[32] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

[33] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer science review*, 3(3):127–149, 2009.

[34] Kohei Nakajima and Ingo Fischer. *Reservoir computing*. Springer, Singapore, 2021.

[35] Katsuma Inoue, Tomoyuki Kubota, Quoc Hoan Tran, Nozomi Akashi, Ryo Terajima, Tempei Kabayama, JingChuan Guan, and Kohei Nakajima. Reservoir computing bootcamp—from python/numpy tutorial for the complete beginners to cutting-edge research topics of reservoir computing. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 36(2), 2026.

[36] Victor Y Pan. How bad are vandermonde matrices? *SIAM Journal on Matrix Analysis and Applications*, 37(2):676–694, 2016.

[37] James Demmel. Accurate singular value decompositions of structured matrices. *SIAM Journal on Matrix Analysis and Applications*, 21(2):562–580, 2000. doi: 10.1137/S0895479897328716. URL `https://doi.org/10.1137/S0895479897328716`.

[38] James Demmel and Plamen Koev. Accurate svds of polynomial vandermonde matrices involving orthonormal polynomials. *Linear algebra and its applications*, 417(2-3):382–396, 2006.

[39] Zlatko Drmac. Svd of hankel matrices in vandermonde-cauchy product form. *Electron. Trans. Numer. Anal*, 44:593–623, 2015.

[40] Dmitry Batenkov, Benedikt Diederichs, Gil Goldman, and Yosef Yomdin. The spectral properties of vandermonde matrices with clustered nodes. *Linear Algebra and its Applications*, 609:37–72, 2021.

[41] Victor Y Pan. Transformations of matrix structures work again. *Linear Algebra and Its Applications*, 465:107–138, 2015.

[42] Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

[43] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=r8H7xhYPwz`.

[44] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

A Usage of large language models

We used large language models to assist nonnative authors with improving the fluency and clarity of the writing. All scientific content was created and validated by the authors.

B Model

S4 [13] is a deep learning architecture designed to efficiently capture long-range dependencies in sequential data. It builds on continuous-time linear SSMs, particularly those formulated in the HiPPO framework [11], and translates them into a trainable and hardware-efficient architecture suitable for modern sequence modeling tasks. S4 begins with the general form of a continuous-time SSM:

$$\dot{\boldsymbol{x}}(t) = \tilde{\boldsymbol{A}}\boldsymbol{x}(t) + \tilde{\boldsymbol{B}}\boldsymbol{u}(t) \qquad \boldsymbol{y}(t) = \tilde{\boldsymbol{C}}\boldsymbol{x}(t) + \tilde{\boldsymbol{D}}\boldsymbol{u}(t), \tag{14}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^N$ is the state, $\boldsymbol{u}(t) \in \mathbb{R}$ is the input, and $\boldsymbol{y}(t) \in \mathbb{R}$ is the output. The matrices $\tilde{\boldsymbol{A}} \in \mathbb{R}^{N\times N}$, $\tilde{\boldsymbol{B}} \in \mathbb{R}^{N\times 1}$, $\tilde{\boldsymbol{C}} \in \mathbb{R}^{1\times N}$, and $\tilde{\boldsymbol{D}} \in \mathbb{R}$ define the system dynamics and output behavior. In a prior study, HiPPO matrices were adopted as $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{B}}$, theoretically realizing the longest memory structure [11]. In S4D, "S4Dinv" and "S4Dlin" were introduced by approximating the eigenvalues of the HiPPO matrices [14].

To implement the SSM on modern computers, the continuous model (Eq. 14) is discretized using the zero-order hold (ZOH) assumption. The input is considered to be held constant over each sampling interval of length Δ. The system can be formulated as

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k \qquad \boldsymbol{y}_k = \tilde{\boldsymbol{C}}\boldsymbol{x}_k + \tilde{\boldsymbol{D}}\boldsymbol{u}_k, \tag{15}$$

where $\boldsymbol{x}_k = \boldsymbol{x}(k\Delta)$, $\boldsymbol{u}_k = \boldsymbol{u}(k\Delta)$, and $\boldsymbol{y}_k = \boldsymbol{y}(k\Delta)$ denote the sampled state, input, and output at discrete time k, respectively. The discrete system matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are derived from the continuous-time matrices as follows: $\boldsymbol{A} = e^{\tilde{\boldsymbol{A}}\Delta}$ and $\boldsymbol{B} = \left(\int_0^\Delta e^{\tilde{\boldsymbol{A}}\tau} d\tau\right)\tilde{\boldsymbol{B}}$. Following prior work on S4 and S4D, Δ is not represented by a single fixed value within a layer but is instead assigned multiple values. For each such Δ, independent SSMs are defined, and within a single layer, all of these SSMs process the input in parallel. As an example, the eigenvalues presented in Fig. 1 are discretized using $\Delta = 0.01$. In Section 3.1 of the main text, we used five eigenvalue realizations. We chose them for the following reasons. The eigenvalues related to the HiPPO matrices, called "S4Dinv" and "S4Dlin,"

were used because they were proposed in previous studies [14] and showed great performance. The definitions are as follows:

$$\text{(S4Dinv)} \quad \lambda_m = -\frac{1}{2} + i\frac{N}{\pi}\left(\frac{N}{2m+1} - 1\right) \qquad \text{(S4Dlin)} \quad \lambda_m = -\frac{1}{2} + i\pi m \ , \tag{16}$$

where λ_m indicates the mth eigenvalue. The eigenvalues named "random" and "linear" are commonly referenced examples in earlier research on the MF. The "step" eigenvalues represent a case in which the "linear" eigenvalues are extended into the complex domain because the eigenvalues of randomly generated matrices always come in complex conjugate pairs when the matrix size N is even. When N is odd, one real eigenvalue is added. Therefore, "step" is more suitable than "linear" for comparison with random eigenvalues.

The S4 architecture extends the basic SSM by enabling both parallelization and deep stacking across multiple layers. In the parallel setting, a sequence of inputs $\{\boldsymbol{u}_k\}_{k=1}^T$ is processed by applying the discretized state-space recurrence to each element independently using fast convolution methods, yielding an output sequence $\{\boldsymbol{y}_k\}_{k=1}^T$. To construct a deep S4 model, multiple such layers are stacked, with each layer consisting of a structured SSM followed by pointwise nonlinear activations and linear transformations. Algorithm 1 shows the detailed flow of one layer.

Mamba extends S4 by making the state space parameters input dependent, allowing the model to selectively retain or discard information based on the input sequence:

$$\dot{\boldsymbol{x}}(t) = \tilde{\boldsymbol{A}}_t \boldsymbol{x}(t) + \tilde{\boldsymbol{B}}_t \boldsymbol{u}(t) \qquad \boldsymbol{y}(t) = \tilde{\boldsymbol{C}}_t \boldsymbol{x}(t) + \tilde{\boldsymbol{D}} \boldsymbol{u}(t) \ , \tag{17}$$

where parameters $\tilde{\boldsymbol{A}}_t$, $\tilde{\boldsymbol{B}}_t$, and $\tilde{\boldsymbol{C}}_t$ are discretized similarly using ZOH. While S4D initializes the recurrent weights λ_m from the eigenvalues of the HiPPO matrix, Mamba adopts the S4D-Real scheme, which initializes the diagonal entries of A as simple integers $\lambda_m = -m$. Algorithm 2 shows the detailed flow of one Mamba layer.

C Memory function of state space models

C.1 Evaluation

The MF of an individual SSM is numerically calculated through the following procedure. First, using Eq. 15, we feed an input time series u_t sampled from a uniform distribution into the model and simulate it for $T_w + T$ time steps, where both T_w and T are sufficiently large. We discard the first T_w steps as a washout period and arrange the remaining T state vectors as rows of $\boldsymbol{X}$. For each delay τ, we construct a corresponding delayed input vector $\boldsymbol{U}_\tau$ of length T. The MFs are obtained by substituting these $\boldsymbol{X}$ and $\boldsymbol{U}_\tau$ into the following formula:

$$M[u_{t-\tau}] = \frac{\boldsymbol{U}_\tau^\top \boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{U}_\tau}{\boldsymbol{U}_\tau^\top \boldsymbol{U}_\tau}, \tag{18}$$

which is equivalent to Eq. 3 [15]. In numerical computation, due to rounding errors and noise, the MF never becomes exactly zero even if the values are already insignificant. To remove insignificant values, we used the surrogate data generated by randomly shuffling the input series in the time direction. The surrogate data preserve the statistical properties of the original data, while temporal dependencies can be eliminated. Replacing the MF target series $\boldsymbol{U}_\tau$ with these surrogate data, we calculated a pseudo MF to evaluate the potential error in the target MF. In practice, we generated 20 surrogate datasets, computed a pseudo MF for each, and took the maximum at each delay τ. We then set target MF values below 1.2 times this maximum to zero. Starting at $\tau = 0$, once an MF value falls below the threshold, we set the MF to zero at that delay and all subsequent delays. These zeros reflect thresholding and truncation, not exact theoretical zeros; truncation may hide later MF increases.

A single SSM layer is generally composed of multiple parallel SSMs. To evaluate an entire layer in terms of MF, it is necessary to take into account the MFs of all constituent SSMs. Given that during forward propagation, the input sequence of the next layer is generated from all the output sequences of the previous layer, we statistically assume that all SSMs within a layer process the same input sequence. We use the maximum individual MF at each delay as a layer summary, not as the capacity of the joint state representation. Here, we additionally provide the average MF as supplementary information (Fig. 5). As is evident, the indicator similarly shows that "S4Dinv" and "S4Dlin" have longer MFs than those obtained with the other eigenvalue configurations.

Algorithm 1 S4D

In the RC setting, the recurrent weights λ and input weights B are fixed.

Require:

Recurrent weights $\lambda \in \mathbb{C}^{H \times N}$
Input weights $B \in \mathbb{C}^{H \times N}$
Output weights $C \in \mathbb{C}^{C \times H \times N}$
Skip weights $D \in \mathbb{R}^{C \times H}$
Output weights $W_{\text{out}} \in \mathbb{R}^{H \times H}$
Input $u \in \mathbb{R}^{\cdots \times L \times H}$

Ensure:

Output sequence $y \in \mathbb{R}^{\cdots \times L \times H}$
for $t = 1$ **to** L **do**
  $x_{:,t+1,:} \leftarrow x_{:,t,:} \odot \lambda + u_t \otimes B$
end for
$z \leftarrow \sum_{n=1}^{N} x_{:,:,n} \odot C_{:,:,n}$
$y \leftarrow 2 \cdot \mathrm{Re}(z)$
$y \leftarrow y + u \otimes D$
$y \leftarrow f(y)$
$y \leftarrow \mathrm{Dropout}(y)$
$y \leftarrow W_{\text{out}}\, y$
return y

Algorithm 2 Mamba

In the RC setting, the parameters λ, W_x, and W_{dt} are fixed.

Require:

Input weights $W_{\text{in}} \in \mathbb{R}^{H \times 2H_{\text{inner}}}$
Convolution weights $W_{\text{conv}} \in \mathbb{R}^{H_{\text{inner}} \times 1 \times K}$
Input weights $W_x \in \mathbb{R}^{H_{\text{inner}} \times (r+2N)}$
Input weights $W_{dt} \in \mathbb{R}^{r \times H_{\text{inner}}}$
Recurrent weights $\lambda \in \mathbb{R}^{H_{\text{inner}} \times N}$
Skip weights $D \in \mathbb{R}^{H_{\text{inner}}}$
Output weights $W_{\text{out}} \in \mathbb{R}^{H_{\text{inner}} \times H}$
Input $u \in \mathbb{R}^{\cdots \times L \times H}$

Ensure:

Output sequence $y \in \mathbb{R}^{\cdots \times L \times H}$
$u_z \leftarrow W_{\text{in}} u$
Split $u_z \rightarrow (u_{\text{all}}, z_{\text{all}})$
$u_{\text{conv}} \leftarrow \mathrm{DepthwiseConv1d}(u_{\text{all}}; W_{\text{conv}})$
$u_{\text{conv}} \leftarrow \phi(u_{\text{conv}})$
$u_{db} \leftarrow W_x u_{\text{conv}}$
Split $u_{db} \rightarrow (dt, B_t, C_t)$
$dt \leftarrow \mathrm{softplus}(W_{dt}\, dt)$
for $t = 1$ **to** L **do**
  $\lambda_t \leftarrow \exp(dt_t \odot \lambda)$; $B_t \leftarrow dt_t \odot B_t$
  $x_{:,t+1,:} \leftarrow x_{:,t,:} \odot \lambda_t + u_{\text{conv},t} \otimes B_t$
end for
$y_t \leftarrow \sum_{n=1}^{N} x_{:,:,n} \odot C_{t,:,n}$
$y \leftarrow y + D \odot u_{\text{conv}}$
$y \leftarrow y \odot \phi(z_{\text{all}})$
$y \leftarrow W_{\text{out}}\, y$
return y

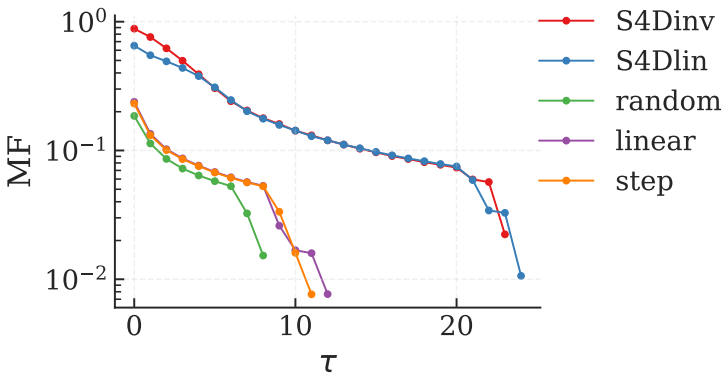


Figure 5: The average MF of five eigenvalue realizations. The horizontal axis is the delay τ of input. The system size N is 32, and the results are obtained by calculating Eq. 3, where $T = 1024$.

C.2 Vandermonde matrix

We adopted the original definition of MF (Eq. 3) instead of the analytical solution of MF (Eq. 5) because of the numerical instability of the Vandermonde matrix, which is a well-studied problem [36–41]. In this study, the issue arises when the MF is computed using the analytical solution of MF under the eigenvalue structures of "S4Dinv" and "S4Dlin." During this computation, the MF exceeds its theoretical range of $[0, 1]$, approaching the precision limits of the machine in both positive and negative values. The computation does not produce reasonable results, suggesting that direct

numerical evaluation of this analytical expression is unreliable for these eigenvalue configurations. Since the MF based on the numerical computation does not exceed its theoretical range and the actual model behavior is expected to follow the numerical procedure, we adopted the solution obtained numerically.

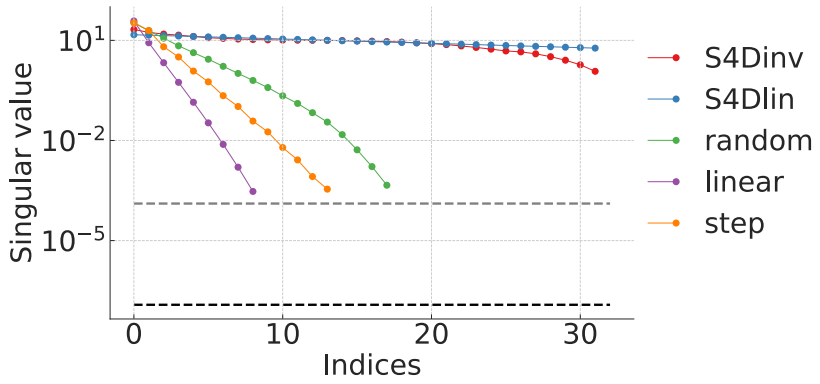


Figure 6: The singular values of matrix $\boldsymbol{V}$ calculated from five eigenvalue realizations. The horizontal axis is an index of singular values. The black line indicates the double-precision machine epsilon ϵ. Singular values below $\max(T, N) \times \max(\sigma) \times \epsilon$ are truncated, where $\max(\sigma)$ is the largest singular value. The maximum threshold calculated from all the realizations is depicted by the gray line. The system size is $N = 32$, and $T = 1024$. As an example, the eigenvalues used here are discretized using $\Delta = 0.01$.

The issue is inherent and may arise in any computation involving $\boldsymbol{V}$. As discussed in the main text, both MF and gradient calculation inevitably involve $\boldsymbol{V}$. This is unavoidable when using linear RNNs, as also shown in the derivations of the original S4 [13]. However, analyzing $\boldsymbol{V}$ is still beneficial. This is supported not only by previous studies demonstrating the consistency between numerical computation and the analytical solution [21] but also by the singular values explaining the long MFs of "S4Dinv" and "S4Dlin" (Fig. 6). From the figure, the structured eigenvalues indicate substantially more singular values above the truncation threshold, thus explaining the results of Fig. 1.

In Section 3.2, we establish the importance of the normalized Vandermonde matrix $\boldsymbol{V}(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}^\top$. To illustrate that the MF can represent matrix $\boldsymbol{V}(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}^\top$ as an indicator, we showed the values in Fig. 7. Though the matrix can be computed directly, the values obtained through this analytical solution exceed reasonable values, as described in the previous paragraph. Therefore, we used a numerical method to calculate the matrix $\boldsymbol{V}(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}^\top$, which is based on the following analytical equation:

$$\boldsymbol{V}(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}^\top = \frac{\hat{\boldsymbol{U}}^\top\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\hat{\boldsymbol{U}}}{T\langle u^2\rangle}, \tag{19}$$

where $\hat{\boldsymbol{U}} = (\boldsymbol{U}_{T-1} \quad \cdots \quad \boldsymbol{U}_0)$, and $\langle u^2\rangle$ is the variance of u. The derivation of this equation follows directly from Appendix D. We evaluated the right-hand side of the equation to obtain a numerical estimate of $\boldsymbol{V}(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}^\top$. The procedure to obtain $\boldsymbol{X}$ and $\boldsymbol{U}_\tau$ follows the method explained in the previous evaluation section.

C.3 Assumptions about random inputs

Our MF-based analysis is limited by the assumption of IID inputs. Despite this limitation, MF provides important insights into input reconstruction and its relationship to learning. With correlated inputs, we can also calculate MF using Eq. 18, which measures how well delayed inputs can be reconstructed from state time series. Here, we derive the expression for $G(\boldsymbol{y})$ in Eq. 10 for an arbitrary input and show that the reconstruction quantity underlying MF appears in the gradient. For an arbitrary input,

$$G(\boldsymbol{y}) = \frac{2}{T}\boldsymbol{X}^\top\left(\boldsymbol{X}\boldsymbol{\theta} - \sum_{k=0}^{T-1}\alpha_k\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{U}_k\right). \tag{20}$$

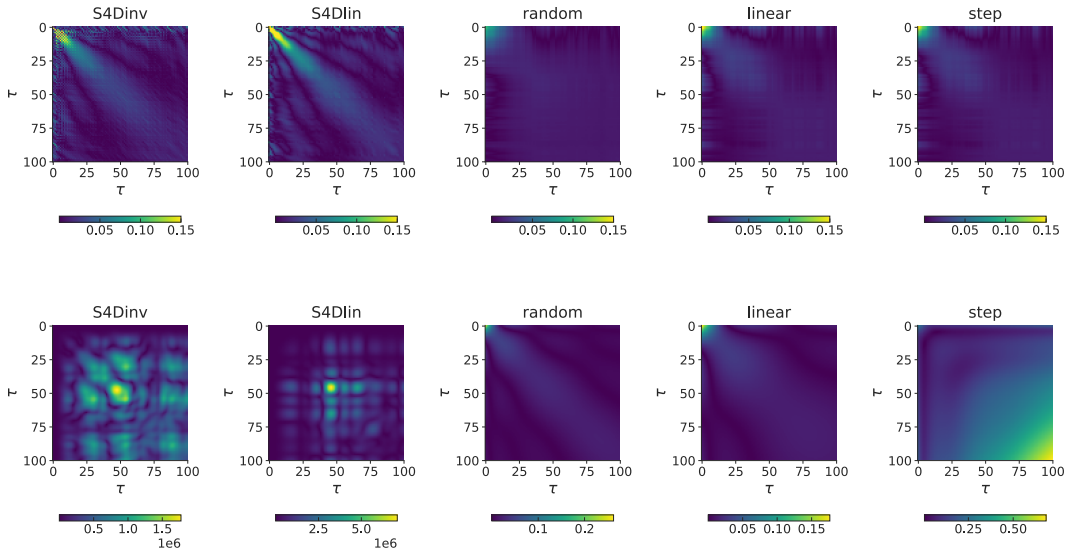


Figure 7: The value of matrix $\boldsymbol{V}(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}^\top$ for five eigenvalue realizations. The upper row shows the values following the numerical calculation. The values in the lower row are directly computed based on the formula $\boldsymbol{V}(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}^\top$. The colorbar ranges in the upper row are the same across all realizations, whereas those in the lower row vary by realization. Both horizontal and vertical axes indicate the delay τ.

Because $\boldsymbol{X}^\top = \boldsymbol{Q}\boldsymbol{V}^\top\hat{\boldsymbol{U}}$ and $\hat{\boldsymbol{U}} = (\boldsymbol{U}_{T-1} \quad \cdots \quad \boldsymbol{U}_0)$, we obtain

$$G(\boldsymbol{y}) = \frac{2}{T}\left(\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{\theta} - \sum_{k=0}^{T-1}\alpha_k\boldsymbol{Q}\boldsymbol{V}^\top\hat{\boldsymbol{U}}\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{U}_k\right), \tag{21}$$

where the term $\hat{\boldsymbol{U}}\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{U}_k$ contains the reconstruction quantity used in the definition of MF (see Appendix D for the derivation). This shows that input reconstruction is related to the gradient even for correlated inputs. However, a delayed input can be reconstructed not only from information retained about the delayed input but also by utilizing correlations between delayed inputs. Thus, a high MF value for correlated inputs does not necessarily indicate that the states retain information about the input at each delay. Such memory arising from input correlations may yield higher reconstruction accuracy than under IID inputs and may allow supervisory information to be reflected more strongly in the readout gradient.

Although MF cannot separately measure these two contributions for correlated inputs, MF measured using IID inputs provides insight into the memory of past input information. Because delayed IID inputs are orthogonal in expectation, their reconstruction cannot utilize correlations between inputs. Even when measured with IID inputs, MF often remains close to 1 over the first several delays (Fig. 1). This indicates that these delayed inputs can be almost completely reconstructed from the states under the IID setting, providing evidence that the states retain information about recent inputs. However, this observation alone does not establish a lower bound on reconstruction accuracy for correlated inputs.

Whether the diagonal components (Fig. 7) also dominate in the correlated-input setting remains unclear. Future work should investigate how input correlations affect the diagonal and off-diagonal contributions to the readout gradient and whether conclusions similar to those obtained under IID inputs hold for correlated inputs.

In addition to these analytical considerations, the theoretical results are consistent with the experimental observations. To examine their practical relevance, we conducted numerical experiments. In the main text, we proved the theoretical equivalence between S4 and S4D from the perspective of MF under IID inputs. The numerical results showed that S4D performs competitively with S4 in most tasks. This observation is consistent with their theoretical equivalence, although comparable task performance does not establish their equivalence in terms of MF beyond the IID setting. Other

theoretical findings presented in the main text are also consistent with the experiments. Therefore, we consider that these theoretical results provide important insights for practical situations and motivate further investigation of their applicability beyond the limitations of this study.

D The Parameter Update Algorithm of State Space Models

Technical details of the analysis of the gradient of SSMs are provided below. To formulate a general parameter update equation, we focus on the lth layer. Using the gradient function $G(\boldsymbol{y})$, we define the parameter update equation for the lth layer as follows:

$$\theta_{t+1}^{l} = F(G, \theta_t^l, \boldsymbol{y}, \epsilon), \tag{22}$$

where θ^l is the parameter for the lth layer, F is a function defined by a certain optimization strategy, and $\boldsymbol{y}$ is the target sequence for the entire model. The function $G = G_l \cdot G_{l+1} \cdots G_L$ is composed of the gradient function G_k from each layer k. In this section, we focus on the gradient G_l in the lth layer and denote it by G, omitting the subscript l.

D.1 The derivation of the update equation with an arbitrary nonlinear readout

In this section, we analytically derive the gradient $G(\boldsymbol{y})$ of one linear SSM layer.

Theorem D.1. *Under the assumptions in Section 3.1 and the idealized input-orthogonality condition stated below, the gradient $G(\boldsymbol{y})$ of a linear SSM layer is given by*

$$G(\boldsymbol{y}) = \frac{2}{T}\boldsymbol{X}^\top \left[f(\boldsymbol{X}\boldsymbol{\theta}) \odot f'(\boldsymbol{X}\boldsymbol{\theta}) - \hat{\boldsymbol{U}}^\top \boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top \bar{\boldsymbol{\alpha}} \right],$$

where $\boldsymbol{X} \in \mathbb{R}^{T\times N}$ is a matrix whose columns consist of state time series, T is the time length, and N is the state size. The matrix $\boldsymbol{V} \in \mathbb{R}^{T\times N}$ represents the Vandermonde matrix composed of the eigenvalues of the internal weight matrix $\boldsymbol{A}$, the vector $\bar{\boldsymbol{\alpha}}$ quantifies the amount of delayed input information contained in the target time series, and $\hat{\boldsymbol{U}} = (\boldsymbol{U}_{T-1} \quad \cdots \quad \boldsymbol{U}_0)$, where the vector $\boldsymbol{U}_k$ is the input series delayed by k time steps. The vector $\boldsymbol{\theta}$ is the parameter vector of the linear readout layer and $f(\cdot)$ is the activation function in the readout layer.

We begin the derivation with the formula $G(\boldsymbol{y}) = \frac{2}{T}\boldsymbol{X}^\top [(f(\boldsymbol{X}\boldsymbol{\theta}) - \boldsymbol{y}) \odot f'(\boldsymbol{X}\boldsymbol{\theta})]$. In S4, the state at time t can be represented as follows:

$$\boldsymbol{x}_t \;=\; \sum_{i=1}^{t} \boldsymbol{A}^{i-1}\boldsymbol{B}u_{t-i+1} = \sum_{i=1}^{t} \boldsymbol{P}\boldsymbol{\Lambda}^{i-1}\boldsymbol{P}^{-1}\boldsymbol{B}u_{t-i+1}, \tag{23}$$

where the initial state is $\boldsymbol{x}_0 = \boldsymbol{0}$, and u_t is the input at time t. We simplified $\boldsymbol{x}_t$ through the eigenvalue decomposition of the internal weight matrix $\boldsymbol{A} = \boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^{-1}$, where $\Lambda = \mathrm{diag}\,(\lambda_1 \quad \lambda_2 \quad \cdots \quad \lambda_N)$, and λ_m is the m-th eigenvalue of $\boldsymbol{A}$. The largest absolute value of λ_m is defined as the spectral radius ρ, which is assumed to satisfy $\rho < 1$. Defining $\boldsymbol{P} = (\boldsymbol{p}_1 \quad \boldsymbol{p}_2 \quad \cdots \quad \boldsymbol{p}_N)$ and $\boldsymbol{P}^{-1} = \begin{pmatrix} \boldsymbol{p}_1'^\top \\ \boldsymbol{p}_2'^\top \\ \vdots \\ \boldsymbol{p}_N'^\top \end{pmatrix}$, we obtain

$$\boldsymbol{x}_t \;=\; \boldsymbol{P}\sum_{i=1}^{t} u_{t-i+1}\mathrm{diag}\begin{pmatrix}\lambda_1^{i-1} & \lambda_2^{i-1} & \cdots & \lambda_N^{i-1}\end{pmatrix}\begin{pmatrix} \boldsymbol{p}_1'^\top\boldsymbol{B} \\ \boldsymbol{p}_2'^\top\boldsymbol{B} \\ \vdots \\ \boldsymbol{p}_N'^\top\boldsymbol{B} \end{pmatrix} \tag{24}$$

$$\;=\; \boldsymbol{P}\sum_{i=1}^{t} \begin{pmatrix} \boldsymbol{p}_1'^\top\boldsymbol{B}\lambda_1^{i-1} \\ \boldsymbol{p}_2'^\top\boldsymbol{B}\lambda_2^{i-1} \\ \vdots \\ \boldsymbol{p}_N'^\top\boldsymbol{B}\lambda_N^{i-1} \end{pmatrix} u_{t-i+1}. \tag{25}$$

Considering the kth delayed state $\boldsymbol{x}_{t-k}$, we obtain

$$\boldsymbol{x}_{t-k} = \boldsymbol{P}\begin{pmatrix} {p'_1}_{\boldsymbol{B}}\boldsymbol{\Lambda}_1^\top \boldsymbol{U}_k \\ {p'_2}_{\boldsymbol{B}}\boldsymbol{\Lambda}_2^\top \boldsymbol{U}_k \\ \vdots \\ {p'_N}_{\boldsymbol{B}}\boldsymbol{\Lambda}_N^\top \boldsymbol{U}_k \end{pmatrix} = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{\Lambda}_1^\top \boldsymbol{U}_k \\ \boldsymbol{\Lambda}_2^\top \boldsymbol{U}_k \\ \vdots \\ \boldsymbol{\Lambda}_N^\top \boldsymbol{U}_k \end{pmatrix},$$

where $\boldsymbol{U}_k = \begin{pmatrix} u_{t-k-(T-1)} & \cdots u_{t-k-(1)} & u_{t-k-(0)} \end{pmatrix}^\top$, $\boldsymbol{\Lambda}_m = \begin{pmatrix} {\lambda_m}^{T-1} & \cdots & {\lambda_m}^1 & {\lambda_m}^0 \end{pmatrix}^\top$, and $\boldsymbol{Q} = \begin{pmatrix} {\boldsymbol{p}'_1}^\top \boldsymbol{B}\boldsymbol{p}_1 & {\boldsymbol{p}'_2}^\top \boldsymbol{B}\boldsymbol{p}_2 & \cdots & {\boldsymbol{p}'_N}^\top \boldsymbol{B}\boldsymbol{p}_N \end{pmatrix}$. As a result, $\boldsymbol{X}$ is described as follows:

$$\boldsymbol{X}^\top = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{\Lambda}_1^\top \boldsymbol{U}_{T-1} & & \boldsymbol{\Lambda}_1^\top \boldsymbol{U}_1 & \boldsymbol{\Lambda}_1^\top \boldsymbol{U}_0 \\ \boldsymbol{\Lambda}_2^\top \boldsymbol{U}_{T-1} & & \boldsymbol{\Lambda}_2^\top \boldsymbol{U}_1 & \boldsymbol{\Lambda}_2^\top \boldsymbol{U}_0 \\ \vdots & \cdots & \vdots & \vdots \\ \boldsymbol{\Lambda}_N^\top \boldsymbol{U}_{T-1} & & \boldsymbol{\Lambda}_N^\top \boldsymbol{U}_1 & \boldsymbol{\Lambda}_N^\top \boldsymbol{U}_0 \end{pmatrix} = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{\Lambda}_1^\top \\ \boldsymbol{\Lambda}_2^\top \\ \vdots \\ \boldsymbol{\Lambda}_N^\top \end{pmatrix}\begin{pmatrix} \boldsymbol{U}_{T-1} & \cdots & \boldsymbol{U}_1 & \boldsymbol{U}_0 \end{pmatrix}. \quad (26)$$

Here, we go back to the formula of the gradient:

$$G(\boldsymbol{y}) = \frac{2}{T}\boldsymbol{X}^\top \left[f(\boldsymbol{X}\boldsymbol{\theta}) \odot f'(\boldsymbol{X}\boldsymbol{\theta}) - \bar{\boldsymbol{y}} \right] \quad (27)$$

$$= \frac{2}{T}\boldsymbol{X}^\top \left[f(\boldsymbol{X}\boldsymbol{\theta}) \odot f'(\boldsymbol{X}\boldsymbol{\theta}) - \boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \bar{\boldsymbol{y}} \right], \quad (28)$$

where $\bar{\boldsymbol{y}} = \boldsymbol{y} \odot f'(\boldsymbol{X}\boldsymbol{\theta})$ represents the label modified by the derivative of the activation function. The covariance matrix $\boldsymbol{X}^\top \boldsymbol{X}$ has the following form:

$$\boldsymbol{X}^\top \boldsymbol{X} = \boldsymbol{Q}\boldsymbol{V}^\top \begin{pmatrix} \boldsymbol{U}_{T-1} & \cdots & \boldsymbol{U}_0 \end{pmatrix} \begin{pmatrix} \boldsymbol{U}_{T-1} & \cdots & \boldsymbol{U}_0 \end{pmatrix}^\top \boldsymbol{V}\boldsymbol{Q}^\top, \quad (29)$$

where $\boldsymbol{V} = \begin{pmatrix} {\lambda_1}^{T-1} & {\lambda_1}^{T-2} & \cdots & {\lambda_1}^1 & 1 \\ {\lambda_2}^{T-1} & {\lambda_2}^{T-2} & \cdots & {\lambda_2}^1 & 1 \\ \vdots & & & & \\ {\lambda_N}^{T-1} & {\lambda_N}^{T-2} & \cdots & {\lambda_N}^1 & 1 \end{pmatrix}^\top$. We assume zero-mean IID inputs with variance $\sigma_u^2 > 0$. Distinct delays are orthogonal in expectation, but not generally in a finite sample. For any fixed finite set of delays, the empirical second-moment matrix converges to σ_u^2 times the identity as the number of observations tends to infinity. The finite-matrix derivations below use the additional idealized condition $\hat{\boldsymbol{U}}^\top \hat{\boldsymbol{U}} = T\sigma_u^2 \boldsymbol{I}_T$, where $\hat{\boldsymbol{U}} = (\boldsymbol{U}_{T-1}, \ldots, \boldsymbol{U}_0)$. This condition is not an exact consequence of finite-sample IID inputs. Using $\begin{pmatrix} \boldsymbol{U}_{T-1} & \cdots & \boldsymbol{U}_0 \end{pmatrix} = \begin{pmatrix} \boldsymbol{U}_{T-1} & \cdots & \boldsymbol{U}_0 \end{pmatrix}^\top$, the covariance matrix $\boldsymbol{X}^\top \boldsymbol{X}$ reduces to the following form:

$$\boldsymbol{X}^\top \boldsymbol{X} = \boldsymbol{Q}\boldsymbol{V}^\top \begin{pmatrix} {\boldsymbol{U}_{T-1}}^\top \\ \vdots \\ {\boldsymbol{U}_0}^\top \end{pmatrix} \begin{pmatrix} \boldsymbol{U}_{T-1} & \cdots & \boldsymbol{U}_0 \end{pmatrix} \boldsymbol{V}\boldsymbol{Q}^\top = T\langle u^2 \rangle \boldsymbol{Q}\boldsymbol{V}^\top \boldsymbol{V}\boldsymbol{Q}^\top, \quad (30)$$

where $\langle u^2 \rangle = \sigma_u^2$ under this idealized condition. Since

$$\boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top = (T\langle u^2 \rangle)^{-1} \begin{pmatrix} \boldsymbol{U}_{T-1} & \cdots & \boldsymbol{U}_0 \end{pmatrix}^\top \boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top \begin{pmatrix} \boldsymbol{U}_{T-1} & \cdots & \boldsymbol{U}_0 \end{pmatrix}, \quad (31)$$

we can observe that the gradient of SSM (Eq. 28) is determined by the matrix $\boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top$. As a result, the gradient $G(\boldsymbol{y})$ is described by

$$G(\boldsymbol{y}) = \frac{2}{T}\boldsymbol{X}^\top \left[f(\boldsymbol{X}\boldsymbol{\theta}) \odot f'(\boldsymbol{X}\boldsymbol{\theta}) - \frac{1}{T\langle u^2 \rangle} \hat{\boldsymbol{U}}^\top \boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top \hat{\boldsymbol{U}}^\top \bar{\boldsymbol{y}} \right], \quad (32)$$

where $\hat{\boldsymbol{U}} = \begin{pmatrix} \boldsymbol{U}_{T-1} & \cdots & \boldsymbol{U}_0 \end{pmatrix}$. Using the fact that a linear SSM performs only linear information processing, all possible outputs of the SSM can be described as the linear combination of delayed input series $\bar{\boldsymbol{y}} = \sum_{k=0}^{T-1} \bar{\alpha}_k \boldsymbol{U}_k = \hat{\boldsymbol{U}}\bar{\boldsymbol{\alpha}}$, where $\{\bar{\alpha}_\tau\}_{\tau=0}^{T-1}$ is a certain series characterizing the modified target sequence $\bar{\boldsymbol{y}}$, and $\bar{\boldsymbol{\alpha}} = \begin{pmatrix} \bar{\alpha}_{T-1} & \cdots & \bar{\alpha}_0 \end{pmatrix}^\top$. By substituting this $\bar{\boldsymbol{y}}$ into $G(\boldsymbol{y})$, the gradient is computed as

$$G(\boldsymbol{y}) = \frac{2}{T}\boldsymbol{X}^\top \left[f(\boldsymbol{X}\boldsymbol{\theta}) \odot f'(\boldsymbol{X}\boldsymbol{\theta}) - \frac{1}{T\langle u^2 \rangle} \hat{\boldsymbol{U}}^\top \boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top \hat{\boldsymbol{U}}^\top \hat{\boldsymbol{U}}\bar{\boldsymbol{\alpha}} \right] \quad (33)$$

$$= \frac{2}{T}\boldsymbol{X}^\top \left[f(\boldsymbol{X}\boldsymbol{\theta}) \odot f'(\boldsymbol{X}\boldsymbol{\theta}) - \hat{\boldsymbol{U}}^\top \boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top \bar{\boldsymbol{\alpha}} \right]. \quad (34)$$

D.2 The update equation with a linear activation function

Next, we consider the linear activation case to simplify the problem and obtain a more intuitive and straightforward understanding.

Lemma D.2. *Under the assumptions on $\boldsymbol{A}$ stated in Section 3.1 and the idealized input-orthogonality condition stated above, the gradient $G(\boldsymbol{y})$ of a linear SSM layer is given by*

$$G(\boldsymbol{y}) = 2\langle u^2\rangle B\boldsymbol{V}^\top(B\boldsymbol{V}\boldsymbol{\theta} - \boldsymbol{V}(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}^\top\boldsymbol{\alpha}),$$

where $\boldsymbol{V} \in \mathbb{R}^{T\times N}$ is the Vandermonde matrix composed of the eigenvalues of the internal weight matrix $\boldsymbol{A}$, T is the time length, and N is the state size. The vector $\boldsymbol{\alpha}$ quantifies the amount of delayed input information contained in the target time series, $\langle u^2\rangle$ is the variance of input u, and B is an arbitrary nonzero constant.

We focus on the gradient before the nonlinear activation layer, where the gradient is described by

$$G(\boldsymbol{y}) \quad = \quad \frac{2}{T}\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}). \tag{35}$$

Following the main text, we consider a typical task, where $\boldsymbol{y}$ is the delayed input series $\boldsymbol{U}_\tau \in \mathbb{R}^T$. The corresponding gradient is

$$G(\boldsymbol{U}_\tau) \quad = \quad \frac{2}{T}(\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{X}^\top\boldsymbol{U}_\tau), \tag{36}$$

where τ is the delay. We have

$$\boldsymbol{X}^\top\boldsymbol{U}_\tau = \boldsymbol{Q}\boldsymbol{V}^\top\begin{pmatrix}\boldsymbol{U}_{T-1}{}^\top\\ \vdots \\ \boldsymbol{U}_0{}^\top\end{pmatrix}\boldsymbol{U}_\tau = \boldsymbol{Q}\boldsymbol{V}^\top\begin{pmatrix}0\\ \vdots\\ 0\\ \boldsymbol{U}_\tau^\top\boldsymbol{U}_\tau\\ 0\\ \vdots\\ 0\end{pmatrix} = T\langle u^2\rangle\boldsymbol{Q}\boldsymbol{V}_\tau, \tag{37}$$

where $\boldsymbol{V}_\tau = (\lambda_1{}^\tau \quad \lambda_2{}^\tau \quad \cdots \quad \lambda_N{}^\tau)^\top$. Combining this result with the nonlinear activation case, we obtain

$$G(\boldsymbol{U}_\tau) = \frac{2}{T}(T\langle u^2\rangle\boldsymbol{Q}\boldsymbol{V}^\top\boldsymbol{V}\boldsymbol{Q}^\top\boldsymbol{\theta} - T\langle u^2\rangle\boldsymbol{Q}\boldsymbol{V}_\tau) = 2\langle u^2\rangle(\boldsymbol{Q}\boldsymbol{V}^\top\boldsymbol{V}\boldsymbol{Q}^\top\boldsymbol{\theta} - \boldsymbol{Q}\boldsymbol{V}_\tau). \tag{38}$$

Following the MF framework introduced in the main text, we simplify S4 to S4D, indicating that $\boldsymbol{P} = \boldsymbol{E}$ and $\boldsymbol{Q} = \mathrm{diag}\,(\boldsymbol{B})$. Using $\boldsymbol{B} = B\boldsymbol{1}$, we can compute the gradient as

$$\begin{aligned} G(\boldsymbol{U}_\tau) \quad &= \quad 2\langle u^2\rangle\boldsymbol{V}^\top\boldsymbol{V}(B^2\boldsymbol{\theta} - B(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}_\tau) &(39)\\ &= \quad 2\langle u^2\rangle B\boldsymbol{V}^\top(B\boldsymbol{V}\boldsymbol{\theta} - \boldsymbol{V}(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}_\tau). &(40)\end{aligned}$$

All the possible gradients can be expressed by substituting the target sequence $\boldsymbol{y} = \sum_{k=0}^{T-1}\alpha_k\boldsymbol{U}_k$ because the linear combination of the delayed input series fully describes the output of the SSM, where $\{\alpha_\tau\}_{\tau=0}^{T-1}$ is a certain series characterizing $\boldsymbol{y}$.

$$\begin{aligned} G(\boldsymbol{y}) \quad &= \quad 2\langle u^2\rangle\boldsymbol{V}^\top\boldsymbol{V}(B^2\boldsymbol{\theta} - B\sum_{k=0}^{T-1}\alpha_k(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}_k) &(41)\\ &= \quad 2\langle u^2\rangle B\boldsymbol{V}^\top(B\boldsymbol{V}\boldsymbol{\theta} - \boldsymbol{V}(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}^\top\boldsymbol{\alpha}). &(42)\end{aligned}$$

D.3 The importance of MF

For example, we consider SGD. A stationary solution for the parameter vector $\boldsymbol{\theta}$ can be analytically derived. $\boldsymbol{\theta}$ is obtained by solving the equation

$$\boldsymbol{\theta} \quad = \quad \boldsymbol{\theta} - \eta_t 2\langle u^2\rangle(B^2\boldsymbol{V}^\top\boldsymbol{V}\boldsymbol{\theta} - B\boldsymbol{V}^\top\boldsymbol{\alpha}). \tag{43}$$

As a result, we obtain $\boldsymbol{\theta}$ and the output $\boldsymbol{X\theta}$ as follows:

$$\boldsymbol{\theta} = \frac{1}{B}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top \boldsymbol{\alpha}, \tag{44}$$

$$\boldsymbol{X\theta} = (\boldsymbol{U}_{T-1} \quad \cdots \quad \boldsymbol{U}_0)^\top \boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top \boldsymbol{\alpha}, \tag{45}$$

where the matrix $\boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top$ appears again, suggesting that MF is inherent in the update algorithm. Actually, this is a reasonable result, as both SGD and MF are derived using the MSE as their loss function. The result obtained here proves the equivalence of the two optimization strategies from a statistical perspective. The role of MF is further revealed by Eq. 11 and Eq. 45, especially by the following part:

$$\boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top \boldsymbol{\alpha}, \tag{46}$$

where $\boldsymbol{\alpha}$ describes the contribution of each delayed input to the target sequence $\boldsymbol{y}$. These coefficients are directly related to how accurately delayed inputs can be reconstructed from $\boldsymbol{y}$. Specifically, substituting $\boldsymbol{X} = \boldsymbol{y}$ into Eq. 18, under the assumed orthogonality and equal variance of the delayed inputs, gives

$$M_y[u_{t-k}] = \frac{\alpha_k^2}{\sum_{j=0}^{T-1} \alpha_j^2}, \qquad \boldsymbol{\alpha} \neq \boldsymbol{0}.$$

Thus, the MF of the target sequence is obtained by normalizing the squared coefficients, whereas $\boldsymbol{\alpha}$ additionally retains their signs and overall scale. The product $\boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top \boldsymbol{\alpha}$ projects these target coefficients onto the column space of $\boldsymbol{V}$. The diagonal entries of this projection matrix are the MF values of the SSM. Consequently, the gradient reflects the interaction between the memory structure of the SSM and the delayed-input content of the target sequence.

D.4 Theoretical results

To interpret the meaning of the gradient, we define a successful parameter update.

Definition D.3. Successful parameter update. We say that a parameter update is successful if the parameter update rule F (Eq. 22) depends on supervisory information. In this paper, supervisory information is represented by $\boldsymbol{\alpha}$ in Lemma D.2.

We use this term if the norm of $\boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top \boldsymbol{\alpha}$ is larger than 0. According to this definition and the gradient formula, the learning mechanisms of SSMs can be partly revealed.

Theorem D.4. *In linear SSMs, a target associated with an exactly zero MF does not contribute to the readout gradient at the current parameters.*

Proof. We focus on the supervisory information α_k associated with the delayed input $\boldsymbol{U}_k$ in the target sequence $\boldsymbol{y} = \sum_{j=0}^{T-1} \alpha_j \boldsymbol{U}_j$. Let

$$\boldsymbol{M} = \boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^{-1}\boldsymbol{V}^\top.$$

As described in Eq. 11, $\boldsymbol{M}$ acts as a filter on the supervisory information $\boldsymbol{\alpha}$, retaining only the component supported by the memory of the SSM. Specifically, it retains the component of the target coefficient vector that can be expressed as a linear combination of the columns of $\boldsymbol{V}$, corresponding to the input-history information represented by the SSM. The filtered supervisory information is therefore $\boldsymbol{M\alpha}$. We show that when the SSM has exactly zero MF at delay k, this filter removes the contribution of α_k.

For each delay j, let $\boldsymbol{e}_j \in \mathbb{R}^T$ be the standard basis vector with 1 at the position corresponding to α_j in $\boldsymbol{\alpha}$ and 0 at every other position. Then

$$\boldsymbol{\alpha} = \sum_{j=0}^{T-1} \alpha_j \boldsymbol{e}_j, \qquad \boldsymbol{M\alpha} = \sum_{j=0}^{T-1} \alpha_j \boldsymbol{M e}_j.$$

The vector $\boldsymbol{M e}_j$ is the column of $\boldsymbol{M}$ corresponding to delay j. Thus, $\alpha_j \boldsymbol{M e}_j$ describes the contribution of the supervisory information α_j after filtering by the SSM's memory.

To separate the contribution of α_k from the others, define

$$\boldsymbol{Y}(\{\alpha_j\}_{j\neq k}) = \sum_{\substack{j=0 \\ j\neq k}}^{T-1} \alpha_j \boldsymbol{M}\boldsymbol{e}_j.$$

With the current SSM parameters fixed, $\boldsymbol{Y}$ depends only on the supervisory coefficients at delays other than k. The filtered supervisory information can therefore be written as

$$\boldsymbol{M}\boldsymbol{\alpha} = \alpha_k \boldsymbol{M}\boldsymbol{e}_k + \boldsymbol{Y}(\{\alpha_j\}_{j\neq k}).$$

It remains to show that the first term vanishes when $M[u_{t-k}] = 0$.

Since $\boldsymbol{M}$ is an orthogonal projector, $\boldsymbol{M}^\top = \boldsymbol{M}$ and $\boldsymbol{M}^2 = \boldsymbol{M}$. Its diagonal entry corresponding to delay k is the MF $M[u_{t-k}]$. Hence,

$$\|\boldsymbol{M}\boldsymbol{e}_k\|^2 = \boldsymbol{e}_k^\top \boldsymbol{M}^\top \boldsymbol{M}\boldsymbol{e}_k = \boldsymbol{e}_k^\top \boldsymbol{M}\boldsymbol{e}_k = M[u_{t-k}].$$

If $M[u_{t-k}] = 0$, this equality gives $\sum_{i=1}^{T} |(\boldsymbol{M}\boldsymbol{e}_k)_i|^2 = \|\boldsymbol{M}\boldsymbol{e}_k\|^2 = 0$, where $(\boldsymbol{M}\boldsymbol{e}_k)_i$ denotes the ith component of $\boldsymbol{M}\boldsymbol{e}_k$. Each term in this sum is nonnegative. The sum can therefore equal zero only if every term is zero, which implies $(\boldsymbol{M}\boldsymbol{e}_k)_i = 0$ for every i. Thus, $\boldsymbol{M}\boldsymbol{e}_k = \boldsymbol{0}$: the entire column corresponding to delay k vanishes.

Consequently, $\alpha_k \boldsymbol{M}\boldsymbol{e}_k = \boldsymbol{0}$ regardless of the value of α_k, and

$$\boldsymbol{M}\boldsymbol{\alpha} = \boldsymbol{Y}(\{\alpha_j\}_{j\neq k}). \tag{47}$$

The memory filter therefore removes the supervisory component associated with delay k, leaving only contributions from the remaining delays.

Substituting this expression into Eq. 11 gives

$$G(\boldsymbol{y}) = 2\langle u^2\rangle B\boldsymbol{V}^\top \left(B\boldsymbol{V}\boldsymbol{\theta} - \boldsymbol{Y}(\{\alpha_j\}_{j\neq k})\right).$$

Since $\boldsymbol{Y}$ is independent of α_k, the readout gradient is also independent of α_k at the current parameters. Thus, supervisory information associated with an exactly zero MF is discarded by the memory filter and does not contribute to the current readout gradient. □

This result motivates initializing the internal weight matrix $\boldsymbol{A}$ to ensure sufficient memory. We next state a necessary condition for predicting delayed inputs.

Lemma D.5. *A nonzero supervisory contribution from the target $y_t = u_{t-k}$ requires nonzero MF at delay k.*

Proof. This follows from Theorem D.4 applied to $y_t = u_{t-k}$ for each k. □

To ensure successful parameter updates, the internal matrix $\boldsymbol{A}$ should satisfy the following condition: the MF must remain sufficiently above zero at every delay k for which α_k is nonzero. To satisfy this condition for an arbitrary task, the MF of an SSM would need to remain sufficiently above zero over an unbounded range of delays.

Theorem D.6. *Memory length takes priority over memory accuracy for a successful parameter update.*

Proof. According to Lemma D.5, an infinitely long memory is required to solve an arbitrary task. Combining this requirement with the trade-off between memory accuracy and memory length (Eq. 4), we conclude that some memory accuracy must be sacrificed to prioritize memory length. □

We next propose two hypotheses about memory changes when recurrent parameters are trained. The first hypothesis is formulated as follows.

Hypothesis D.7. Memory decreases as learning proceeds. The decrease in memory is defined as

$$M[u_{t-\tau}]_k > M[u_{t-\tau}]_{k+1}, \tag{48}$$

where k indicates the number of steps for parameter updates.

This occurs because the memory architecture can be adjusted in accordance with the capacity required by the task. Since MC is analytically constant, an increase in the MF at some delays must be accompanied by a decrease at others. If this phenomenon impairs generalization performance, it suggests overfitting. Note that if learning increases the system rank, this trade-off does not arise, and increasing memory is always beneficial.

To formulate the second hypothesis, we define the term "unintended extension of memory."

Definition D.8. Unintended extension of memory. Assume that the input information $u_{t-\tau_1}$ is necessary to predict the label information y_t in a task and that for $u_{t-\tau_1}$, the MF satisfies $M[u_{t-\tau_1}] > 0$. Then, consider the MF for a more distant past input $u_{t-\tau_2}$ with $\tau_2 > \tau_1$, i.e., $M[u_{t-\tau_2}] \geq 0$. Unintended extension refers to the situation in which, during training, the model tries to improve $M[u_{t-\tau_1}]$, and as a side effect, $M[u_{t-\tau_2}]$ also increases.

Following this definition, the second hypothesis is formulated.

Hypothesis D.9. Unintended extension of memory occurs as learning proceeds.

The two effects can coexist, and their net impact on performance must be assessed empirically. These considerations motivate our experiments in the RC setting. These experiments determine which of the two predicted effects dominates. Specifically, regarding the unintended extension of memory, we focus on a particularly important case where $M[u_{t-\tau_2}] = 0$ before training. We empirically confirm in Fig. 9 that in this case, MF extension occurs (Section 3.3).

D.5 Extension to a broader range of loss functions

Theoretical results are derived from Lemma D.2, which is restricted to the linear readout. However, the same results can be obtained from the nonlinear case as shown in Theorem D.1. This may enable us to extend the theoretical results to a broader class of loss functions. As an example, we explain how our analysis relates to the cross-entropy loss, which is used in our numerical experiments on LRA tasks and language modeling. According to the definition, $L_{\text{MSE}} = \langle (\hat{y}_t - y_t)^2 \rangle = \langle \hat{y}_t^2 \rangle - 2\langle \hat{y}_t y_t \rangle + \langle y_t^2 \rangle$, where $\hat{y}_t = f(\boldsymbol{\theta}^\top \boldsymbol{x}_t)$. If we take $f = \log$ with $\boldsymbol{\theta}^\top \boldsymbol{x}_t > 0$, then

$$L_{\text{MSE}} = \langle \hat{y}_t^2 \rangle + 2\langle \left(-y_t \log(\boldsymbol{\theta}^\top \boldsymbol{x}_t)\right) \rangle + \langle y_t^2 \rangle = \langle \hat{y}_t^2 \rangle + 2\text{CE}\left(y_t, \boldsymbol{\theta}^\top \boldsymbol{x}_t\right) + \langle y_t^2 \rangle, \tag{49}$$

where $\text{CE}(y_t, z_t) = -\langle y_t \log z_t \rangle$ is a log-loss term. Here, the term $\langle y_t^2 \rangle$ can be dropped as it is irrelevant: it does not depend on the parameter vector $\boldsymbol{\theta}$ and therefore has zero gradient with respect to $\boldsymbol{\theta}$. Thus, this result concerns the composite loss L_{eff}, not standard cross-entropy alone.

$$L_{\text{eff}} = \langle \hat{y}_t^2 \rangle + 2\text{CE}\left(y_t, \boldsymbol{\theta}^\top \boldsymbol{x}_t\right). \tag{50}$$

E Experimental Details

In this section, we provide the detailed methods and additional results of the experiments.

E.1 Long Range Arena

The LRA benchmark includes ListOps, byte-level text classification (Text), byte-level document retrieval (Retrieval), image classification on sequences of pixels (Image), a task to determine whether a valid path exists between two points from pixel sequences (Pathfinder), and another more difficult Pathfinder task with a longer sequence (PathX).

The models are constructed following the official GitHub repository[2] provided by the authors of S4 [13]. According to the result that S4 and S4D are equivalent, we focus on the S4D model, which has lower computational cost. The hyperparameters are described in Table 2. Here, `n_layers`,

[2] `https://github.com/state-spaces/s4/tree/simple`

d_model, and n_ssm denote the number of layers, the number of SSMs per layer (H in Algorithm 1), and the number of distinct matrices $\boldsymbol{A}$, respectively. Next, lr, lr_layer, and lr_dt indicate the learning rate for the entire model, the learning rate for each layer, and the learning rate for Δ, respectively. Finally, dt_min and dt_max correspond to the minimum and maximum values of Δ, respectively. In the implementation, $N = \texttt{d_state}/2 = 32$ counts complex state variables. The activation function is defined as $f =$ GELU. In practice, a bidirectional operation was introduced. With this operation, the SSM kernel outputs two time series: the sequence corresponding to the original ordering of time steps and an additional time-reversed sequence. As a result, the linear readout $\boldsymbol{W}_{out}$ receives an input sequence that is twice the length of the layer input time series. In addition, after the linear readout layer, post activation $g =$ GLU was introduced. Cross-entropy was employed as the loss function in the final layer. Our results confirm that the findings are consistent with the theoretical outcomes obtained using MSE. Table 3 compares model sizes before and after applying the RC setting.

In Fig. 3, we showed the supremum MF and linear readout weights of one SSM layer when the training of eigenvalues was allowed. For completeness, we additionally provide the results of all layers (Fig. 9 and Fig. 11) and the average MF before and after training (Fig. 10). As with the supremum MF, the MF for the structured eigenvalues "S4Dinv" and "S4Dlin" exhibits longer memory than the MFs obtained with the other eigenvalue configurations. To reveal the reason for the modest changes in the MFs, we show the eigenvalues before and after training (Fig. 8). At initialization, the eigenvalue magnitudes are close to 1, and this characteristic is maintained even after training. Excluding PathX, each training and evaluation session was completed in approximately five hours. For PathX, it took about 20 hours. In all tasks, the training was performed using two NVIDIA A100 Tensor Core GPUs, each with 40 GB of memory.

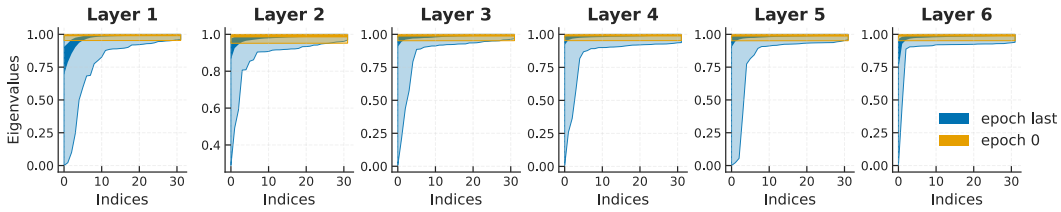


Figure 8: Eigenvalue magnitudes of S4Dinv across six layers before and after training. The horizontal axis is the index. Color indicates the epoch. The dark shading represents the interquartile range between Q1 and Q3, while the light shading indicates the range from the minimum to the maximum. The system size N is 32. As an example, the case of the Pathfinder task is shown.

Table 2: Hyperparameters of the model

Parameter	ListOps	Text	Retrieval	Image	Pathfinder	PathX
n_layers	8	6	6	6	6	6
d_model	128	256	256	512	256	256
d_state	64	64	64	64	64	64
n_ssm	128	256	256	2	256	256
lr	0.01	0.01	0.01	0.01	0.004	0.0005
lr_layer	0.001	0.001	0.001	0.001	0.001	0.0005
lr_dt	0.01	0.01	0.01	0.001	0.001	0.0005
dt_min	0.001	0.001	0.001	0.001	0.001	0.0001
dt_max	0.1	0.1	0.1	0.1	0.1	0.01
dropout	0.	0.	0.	0.1	0.	0.
weight_decay	0.05	0.05	0.05	0.05	0.03	0.05
batch_size	50	16	32	50	64	16
max_epoch	40	32	20	200	200	50

Table 3: Comparison of trainable parameter size before and after applying the RC setting to each task. From left to right, the columns show the trainable parameter size in the RC setting, the trainable parameter size without the RC setting, and the reduction ratio achieved by applying the setting.

	Memory occupied by trainable parameters (Megabytes)		
Task	Proposed method	Conventional method	Reduction (%)
ListOps	1.608	1.874	14.19
Text	4.104	4.504	8.88
Retrieval	5.117	5.516	7.23
Image	14.217	14.233	0.11
Pathfinder	3.969	4.368	9.13
PathX	3.969	4.368	9.13

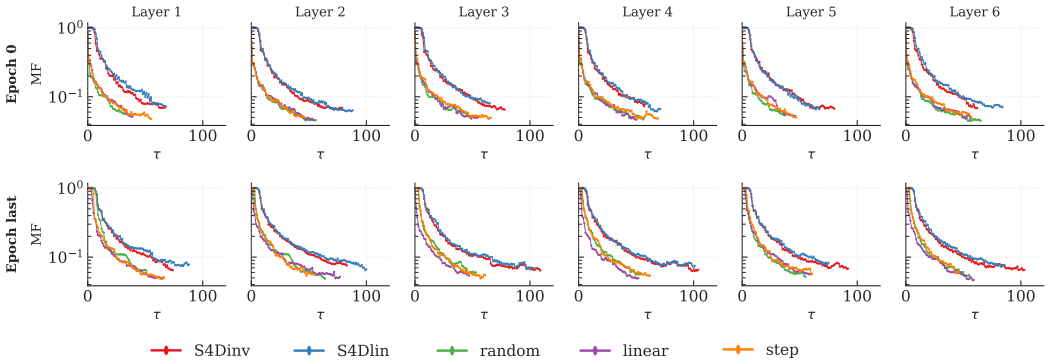


Figure 9: The supremum MF of each layer before and after training. The top (bottom) panels show the MFs before (after) training. From left to right, each column represents one layer. The horizontal (vertical) axes show the input delay τ (MF). The five colors indicate the eigenvalue realizations. The state size N is 32, and the results are obtained by calculating Eq. 3 with $T = 1024$. As an example, we show the results for the Pathfinder task.

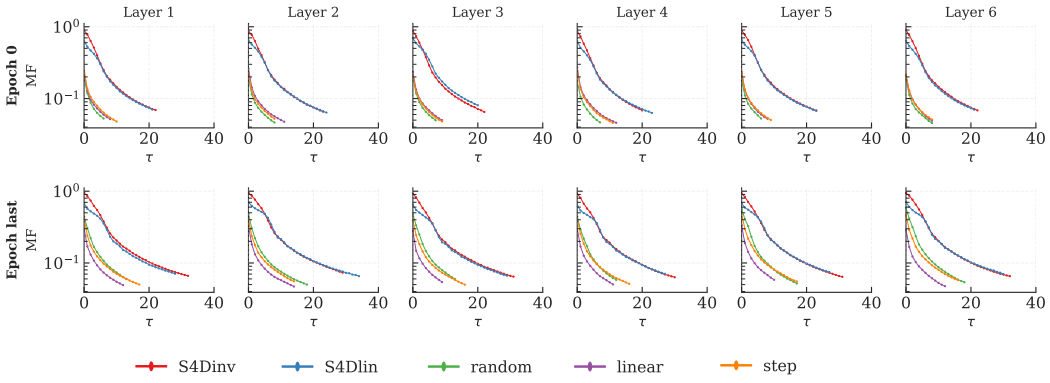


Figure 10: The average MF of each layer before and after training. The upper (lower) panels show the MFs before (after) training. From left to right, each column corresponds to a layer. The horizontal (vertical) axes show the input delay τ (MF). The five colors indicate the eigenvalue realizations. The state size N is 32, and the results are obtained by calculating Eq. 3, where $T = 1024$. As an example, we show the case of the Pathfinder task.

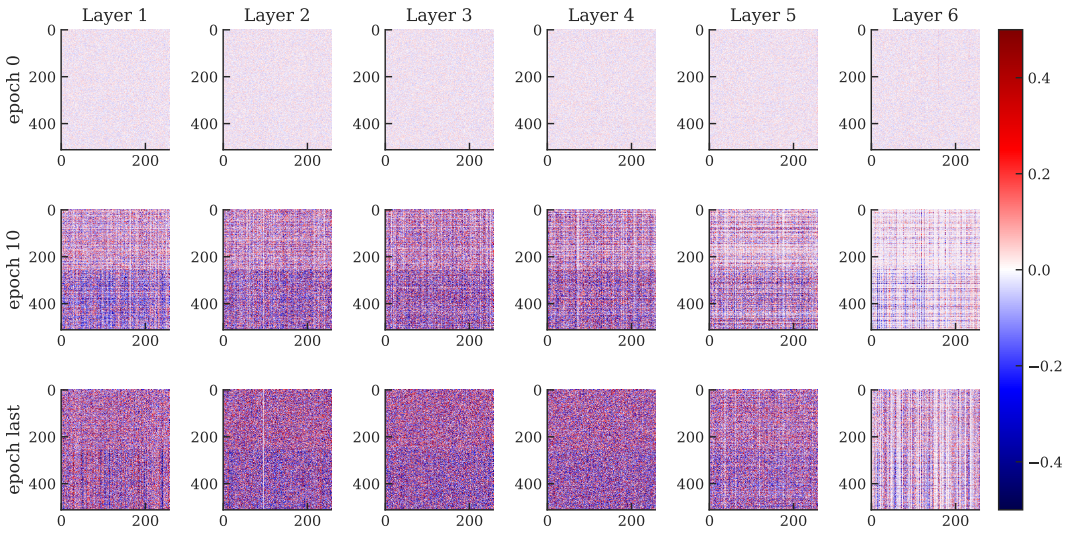


Figure 11: The linear readout weights $\boldsymbol{W}_{out}$ at each layer and epoch. From top to bottom, the rows correspond to epochs 0, 10, and the final epoch. From left to right, the columns show the weights of layers 1 through 6. Following Fig. 9, we present an example from the Pathfinder task.

Furthermore, the loss and accuracy during the training and test phases are provided as supplementary materials.

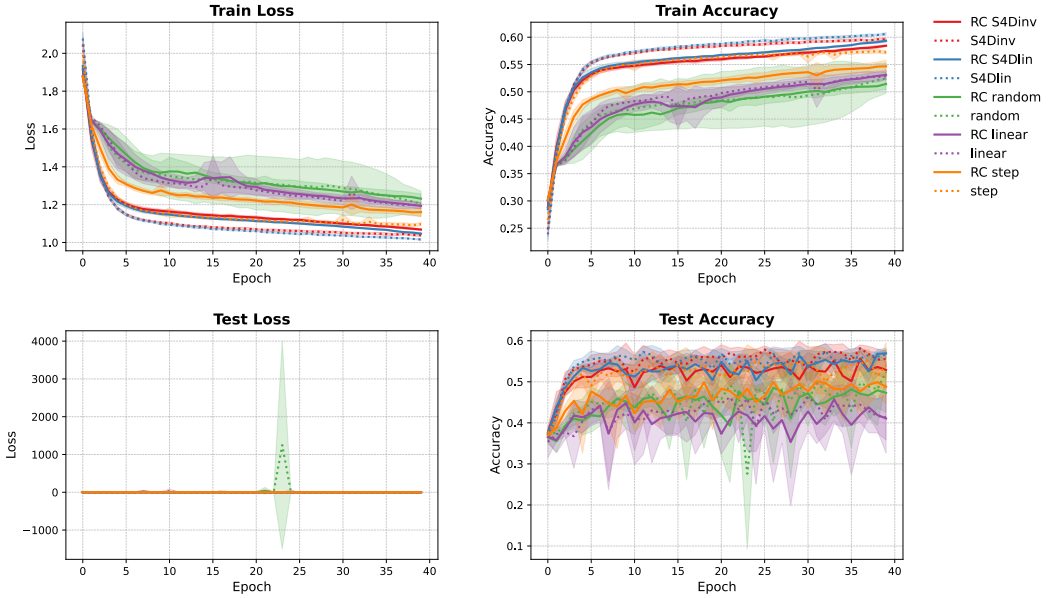


Figure 12: Training and test loss and accuracy on the ListOps task. The upper (lower) row shows training (test) metrics. Each column corresponds to a different metric. The horizontal axes indicate epochs. The colors indicate eigenvalue realizations. Solid lines represent models in the RC setting, while dotted lines represent models with trainable eigenvalues.

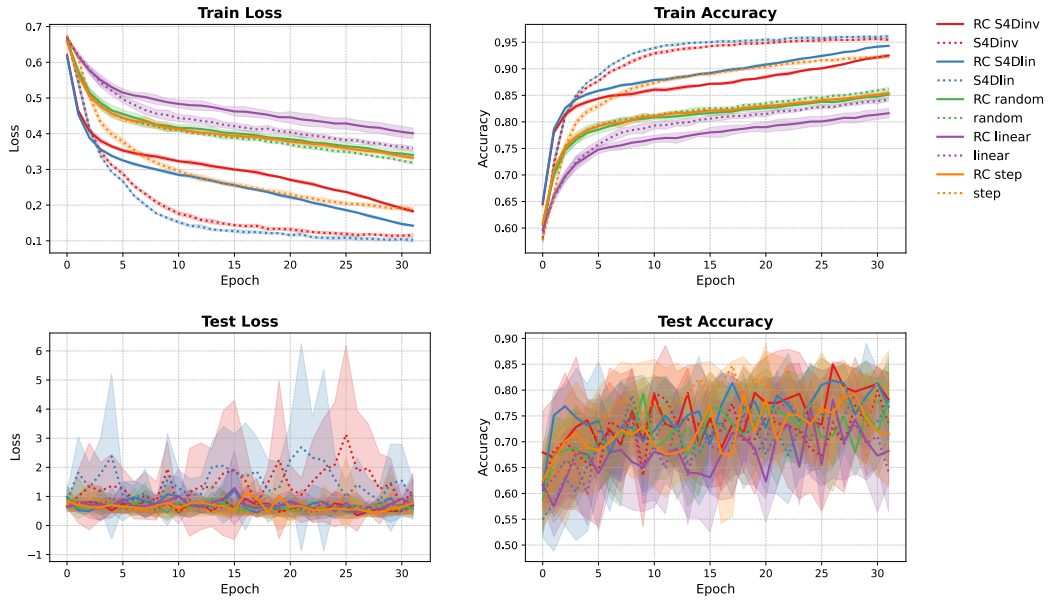


Figure 13: Training and test loss and accuracy on the Text task. The upper (lower) row shows training (test) metrics. Each column corresponds to a different metric. The horizontal axes indicate epochs. The colors indicate eigenvalue realizations. Solid lines represent models in the RC setting, while dotted lines represent models with trainable eigenvalues.

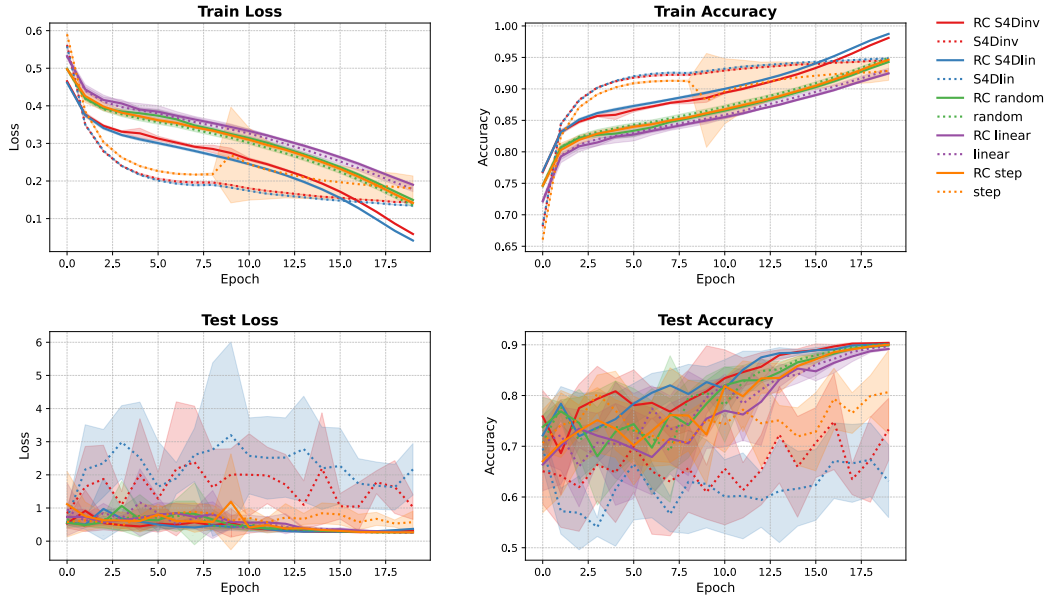


Figure 14: Training and test loss and accuracy on the Retrieval task. The upper (lower) row shows training (test) metrics. Each column corresponds to a different metric. The horizontal axes indicate epochs. The colors indicate eigenvalue realizations. Solid lines represent models in the RC setting, while dotted lines represent models with trainable eigenvalues.

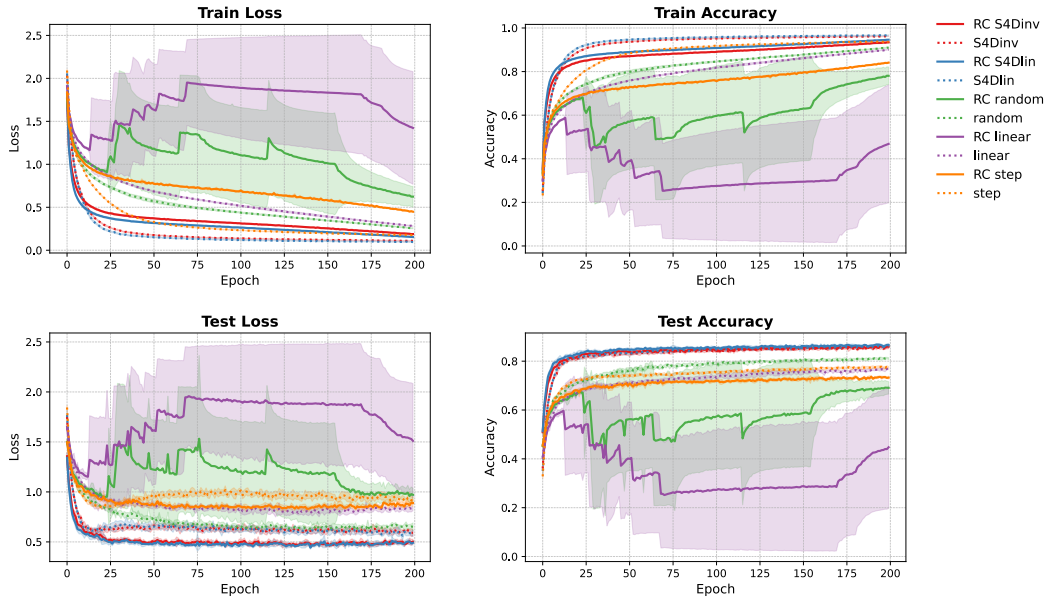


Figure 15: Training and test loss and accuracy on the Image task. The upper (lower) row shows training (test) metrics. Each column corresponds to a different metric. The horizontal axes indicate epochs. The colors indicate eigenvalue realizations. Solid lines represent models in the RC setting, while dotted lines represent models with trainable eigenvalues.

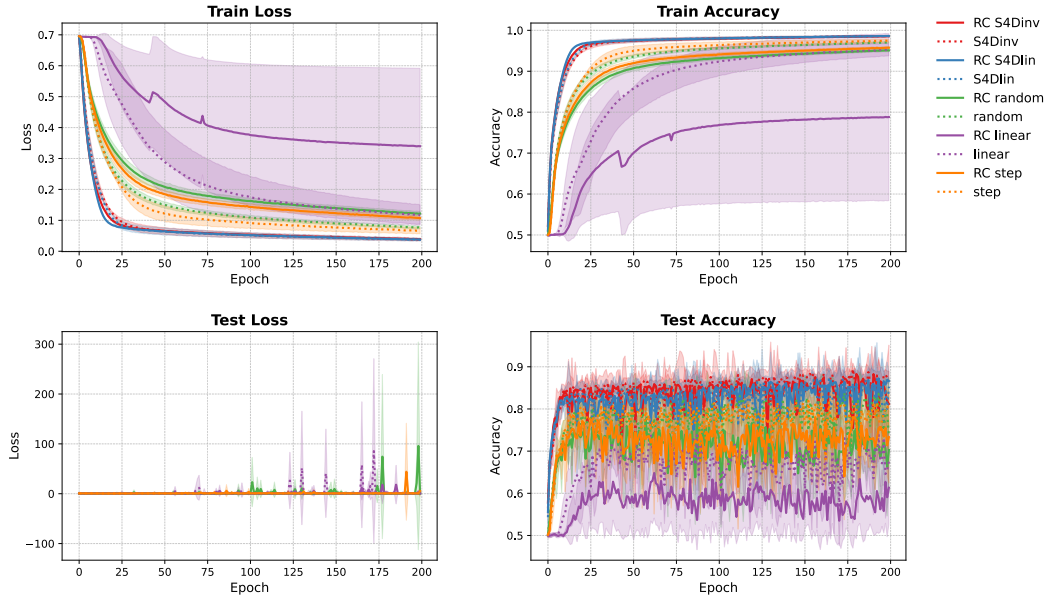


Figure 16: Training and test loss and accuracy on the Pathfinder task. The upper (lower) row shows training (test) metrics. Each column corresponds to a different metric. The horizontal axes indicate epochs. The colors indicate eigenvalue realizations. Solid lines represent models in the RC setting, while dotted lines represent models with trainable eigenvalues.

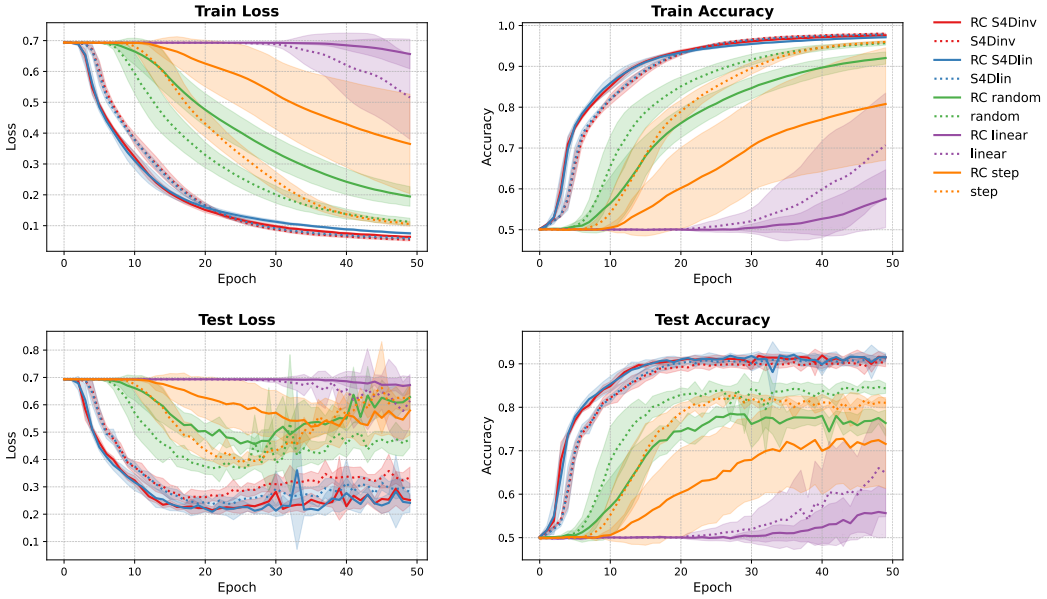


Figure 17: Training and test loss and accuracy on the PathX task. The upper (lower) row shows training (test) metrics. Each column corresponds to a different metric. The horizontal axes indicate epochs. The colors indicate eigenvalue realizations. Solid lines represent models in the RC setting, while dotted lines represent models with trainable eigenvalues.

E.2 Language modeling

S4D and Mamba in the RC setting were evaluated on a language modeling task. Both models were trained to predict the next token given the input sequence. All language modeling experiments used the `sample-10BT` subset of FineWeb [42], tokenized using the GPT-2 tokenizer with a sequence length of 2048 tokens. In the main text, we presented the results of Mamba (Fig. 4). We additionally report the corresponding results for S4D in Fig. 18. Similarly, under the RC setting, the loss was observed to be comparable to or even lower than that of full training. Moreover, following S4D, we compare the MF of Mamba before and after training at each layer (Fig. 19).

In the RC setting of S4D, the eigenvalues of the recurrent weights and the input weights are fixed, whereas in Mamba, W_x, W_{dt}, and λ in Algorithm 2 are fixed. The language model was constructed by stacking multiple blocks, each consisting of a Mamba layer and a two-layer perceptron. The S4D-based model was implemented by replacing each Mamba layer with an S4D layer. For each layer, the hyperparameters were set to the default values proposed in prior work. Both models had 12 layers, with $H = 768$ for Mamba and $H = 1536$ for S4D. The global batch sizes, including gradient accumulation across four GPUs, were 49,152 tokens per optimizer update for Mamba and 196,608 for S4D. We used Adam with $(\beta_1, \beta_2) = (0.9, 0.95)$ and weight decay 0.01. We used a cosine learning-rate schedule with warmup ratios of 0.05 for Mamba and 0.02 for S4D. The plotted learning rates were $\{0.001, 0.005, 0.01\}$ for Mamba and $\{0.003, 0.005, 0.01\}$ for S4D. We searched over the learning rates listed above and selected configurations with the lowest final training loss. For all result figures, we report the full training curves without truncation or selective cropping. The reported loss is the training loss. In the language modeling task, each training run required approximately three days. For all models, training was performed using four NVIDIA A100 Tensor Core GPUs, each with 40 GB of memory.

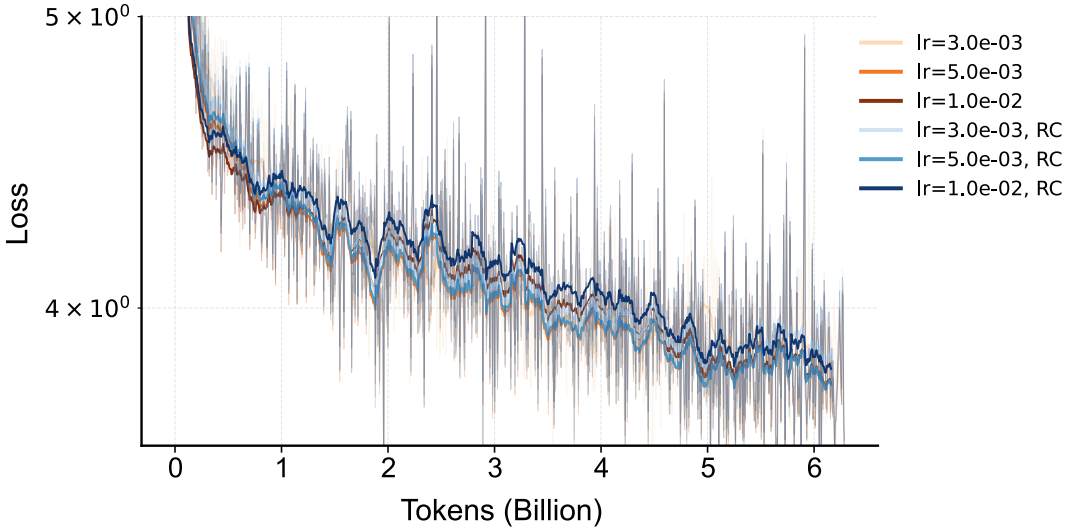


Figure 18: The loss dynamics of S4D in the RC setting. The task is language modeling. The horizontal axis indicates the number of tokens used for training. The solid line represents the result smoothed using a 20-step sliding window, while the thin line corresponds to the unsmoothed result. The same smoothing procedure is used in Fig. 4.

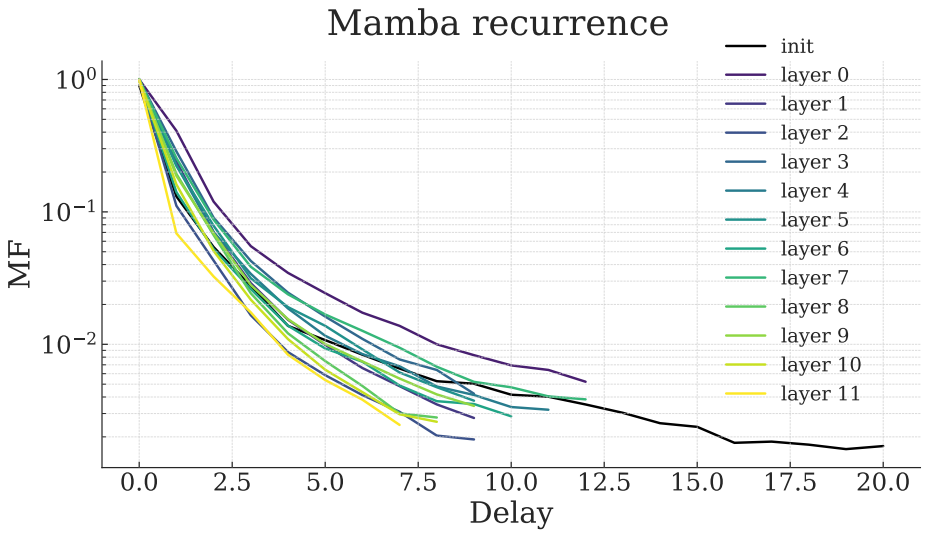


Figure 19: MF of Mamba at each layer before and after training. The horizontal (vertical) axis shows the delay (MF), while "init" represents the result of a layer before training.

E.3 Licenses

We use several existing assets with the following licenses:

- S4 / S4D: Apache License 2.0 (`https://github.com/state-spaces/s4`)
- Mamba: Apache License 2.0 (`https://github.com/state-spaces/mamba`)
- LRA: Apache License 2.0 (`https://github.com/google-research/long-range-arena`)
- FineWeb dataset: ODC-BY (`https://huggingface.co/datasets/HuggingFaceFW/fineweb`)
- Evaluation of MF and IPC: MIT License (`https://github.com/kubota0130/ipc`)

All assets are used in compliance with their respective licenses.

F Further Discussion

F.1 Related research

Related research topics include vanishing gradient problems [4–6]. These studies focused on multi-layer networks, in which gradients accumulate multiplicatively as errors propagate through the layers. For RNNs in particular, the problem is tied to the traditional BPTT algorithm, in which gradients propagate backward through the recurrent time steps. In the model examined in this study, the notion of "multiple layers" can be understood in the two ways described above. One interpretation (a) is to deepen the model by stacking several distinct SSMs along the depth direction. From this perspective, the number of layers corresponds to the number of different SSM modules. The other interpretation (b) focuses on the temporal evolution within a single SSM. In this case, the number of layers is regarded as the number of time-propagation steps. We focused on the latter interpretation (b) in this study. Here, we explain how the vanishing gradient problem arises in this case and how the results obtained in this study can be used to understand it. To achieve this, we first revisit the gradient expressions. For a target time series $\mathbf{y}$, the gradient with respect to the linear readout weights under MSE is given by the expression

$$G(\boldsymbol{y}) \quad = \quad \frac{2}{T}\boldsymbol{X}^\top \boldsymbol{e}, \tag{51}$$

where $\boldsymbol{e} = (\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y})$ denotes the error time series. By simplifying this expression analytically, we obtain the expression

$$G(\boldsymbol{y}) \quad = \quad 2\langle u^2\rangle B\boldsymbol{V}^\top \bar{\boldsymbol{e}}, \tag{52}$$

where $\bar{\boldsymbol{e}} = (B\boldsymbol{V}\boldsymbol{\theta} - \boldsymbol{V}(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}^\top\boldsymbol{\alpha})$, and $\boldsymbol{V} = \begin{pmatrix} {\lambda_1}^{T-1} & {\lambda_1}^{T-2} & \cdots & {\lambda_1}^{1} & 1 \\ {\lambda_2}^{T-1} & {\lambda_2}^{T-2} & \cdots & {\lambda_2}^{1} & 1 \\ \vdots & & & & \\ {\lambda_N}^{T-1} & {\lambda_N}^{T-2} & \cdots & {\lambda_N}^{1} & 1 \end{pmatrix}^\top$. Based on these formulas, we discuss how the error information decays.

Here, the Vandermonde matrix $\mathbf{V}$ characterizes how the error information decays through BPTT. As shown in Section 3.2, the decay of this matrix can be evaluated by the MF. Considering the case in which the input weight is removed, the state time series generated by injecting input u_t is given by $B\boldsymbol{V}^\top \begin{pmatrix} {\boldsymbol{U}_{T-1}}^\top \\ \vdots \\ {\boldsymbol{U}_0}^\top \end{pmatrix}$ because $\boldsymbol{Q}$ is reduced to $B\mathbf{1}$. Replacing the input sequence with the error sequence yields a state trajectory of the form $\boldsymbol{Q}\boldsymbol{V}^\top\bar{\boldsymbol{e}}$, which is exactly the term in Eq. 52. By evaluating how accurately the error information can be reconstructed from the state time series, we can assess how much the error information decays through BPTT. As this is exactly the definition of the MF, the decay of error information can be considered captured by the MF.

This shows that a single exponential decay does not generally characterize the memory profile. For example, when the delay is small, the MF remains close to 1, as shown in Fig. 1. Accordingly, using the MF may enable a more accurate evaluation of how error information decays.

In this study, the error $\bar{\boldsymbol{e}}$ was further decomposed as a function of the input. Beyond the previously discussed decay of error signals, we also clarified the loss of input information. As a result, we revealed a mechanism of loss of supervisory signals that cannot be explained solely by focusing on the decay of error signals, as described in the main text. This result suggests that the gradient may not depend on the input even when the error $\bar{\boldsymbol{e}}$ is nonzero (Eq. 52), in which case, supervisory information does not guide the parameter update direction. This occurs because $\boldsymbol{V}(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}^\top\boldsymbol{\alpha} = 0$ may hold, while generally $B\boldsymbol{V}\boldsymbol{\theta} \neq 0$. Here, $\boldsymbol{\alpha}$ represents the supervisory signals, and, similarly to the formulation of error signal decay, the MF represented by the matrix $\boldsymbol{V}(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}^\top$ continues to appear. As $\boldsymbol{\alpha}$ is associated with past inputs, the term $\boldsymbol{V}(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}^\top\boldsymbol{\alpha}$ represents the inner product of the memory supplied by the SSM and the memory demanded by the supervisory signals. Therefore, the condition $\boldsymbol{V}(\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{V}^\top\boldsymbol{\alpha} = 0$ indicates that there is no overlap between these memories. The dependence of recurrent learning on the readout weights follows directly from BPTT. Consider a linear RNN $\boldsymbol{x}_t = \boldsymbol{A}\boldsymbol{x}_{t-1} + \boldsymbol{B}u_t$ with $\hat{y}_t = \boldsymbol{\theta}^\top\boldsymbol{x}_t$ and a fixed initial state. For

$L_{\mathrm{MSE}} = T^{-1}\sum_{t=1}^{T}(\hat{y}_t - y_t)^2$, let $e_t = \hat{y}_t - y_t$ and $\boldsymbol{\delta}_t = \partial L_{\mathrm{MSE}}/\partial \boldsymbol{x}_t$. BPTT gives

$$\boldsymbol{\delta}_t = \frac{2}{T} e_t \boldsymbol{\theta} + \boldsymbol{A}^\top \boldsymbol{\delta}_{t+1}, \qquad \boldsymbol{\delta}_{T+1} = \mathbf{0}, \tag{53}$$

$$\nabla_{\boldsymbol{A}} L_{\mathrm{MSE}} = \sum_{t=1}^{T} \boldsymbol{\delta}_t \boldsymbol{x}_{t-1}^\top. \tag{54}$$

Thus, the current readout weights determine how output errors enter the recurrent gradient. In particular, $\boldsymbol{\theta} = \mathbf{0}$ implies $\nabla_{\boldsymbol{A}} L_{\mathrm{MSE}} = \mathbf{0}$. This explicitly establishes a coupling between readout and recurrent learning.

On the other hand, MC indicates a trade-off between long-term memory and memory accuracy. Moreover, prior work has shown that longer memory intensifies output variations induced by small parameter changes. This suggests a trade-off between long-term memory and learning stability. Based on these facts, several directions for future research can be considered. One is to identify the optimal memory length under these trade-offs. Another is to develop learning mechanisms that can adapt the memory length to its optimal value during training. We believe that the results of this study contribute to these directions.

F.2 Numerical experiments

In this study, we used the same hyperparameters as in prior work [13], changing only the random seed. We consider the hyperparameter settings presented in prior work to be close to optimal and therefore believe that further performance improvements are extremely difficult under the conventional full-parameter training framework. However, some of our results for S4D variants differed from those reported in the S4D paper [14]. Difficulties reproducing the S4 results have also been reported on GitHub. It has been pointed out that such irreproducibility may stem from differences in hardware or software, such as GPU or CPU models, or PyTorch versions. In our environment, the CPU differed from that used in the original implementation. We ran multiple trials with different random seeds, but some results still differed from those reported in prior work. We consider such discrepancies to be within a reasonable range, consistent with discussions on GitHub. While it is possible that parameter retuning would be necessary in a different environment, we did not pursue this, as it falls outside the main focus of our study.

F.3 Applicability to broader models

Our numerical results support the effectiveness of the RC setting in Mamba (Appendix E.2), suggesting that the results may be applicable to a wider range of models, including Mamba2 [2] and gated delta networks [43]. However, nonlinear SSMs must be treated carefully because they include nonlinear information processing in addition to the linear processing that can be evaluated by MFs. Beyond the involvement of nonlinear processing, nonlinear operations may alter the components of linear information processing. As IPC enables us to comprehensively investigate nonlinear information processing, it may broaden the findings of this study and uncover new insights.

Since causal linear Transformers admit an RNN formulation [44], they provide a promising basis for extending our MF-based analysis beyond SSMs. Such an extension requires accounting for their matrix-valued memory states, input-dependent operations, and non-decaying recurrence, and remains an important direction for future work.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately describe the backgrounds of the research area, present issues, and how our study contributes to it.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: In the section titled Discussion and Conclusion, we examine in detail the limitations inherent in the models considered in this work, as well as the aspects of the learning dynamics that remain unexplored in this study.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: In the Results section, we provide a detailed description of the assumptions and conditions that define the scope of our analysis. Furthermore, all necessary information is explicitly stated to ensure that the derivations presented in the main text are fully understandable.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: To ensure reproducibility of the experimental results, all derived equations are explicitly stated. For experiments conducted with reference to prior work, all relevant parameters are specified, and the original sources are clearly cited.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: To ensure reproducibility of the figures and tables presented in the main text, we provide detailed implementation descriptions and the underlying equations. In addition, all experiments can be reproduced by following the procedures described in the paper, using the cited references and the mathematical formulations provided in the manuscript.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: To facilitate reproducibility, all experimental settings used in this study are clearly specified. In particular, since the study involves the use of specific eigenvalue configurations of certain matrices, we reference the original literature where their definitions are introduced and visualize the actual values used. The captions of all presented figures include detailed information about the corresponding experimental parameters.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: The paper reports error bars appropriately and clearly defines them (e.g., standard deviation and standard error). In addition, it provides sufficient information about statistical significance, such as repeated runs and variance across trials, which supports the reliability of the experimental results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The computational environment used in this study is explicitly specified in the Experimental Details section.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research provides a method with a theoretical explanation on how to improve the performance of the modern frequently used machine learning. It does not involve human subjects or obviously ethically sensitive applications, and we consider it conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.

- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: This paper identifies parameters that are fundamentally essential to SSMs and proposes a method for simplifying existing models. This approach contributes to improved computational efficiency by reducing the number of training parameters and training steps. Moreover, the method has the potential to enhance the performance of SSMs, which are widely used in various applications such as natural language processing and computer vision. Given its application to a class of models that has recently attracted substantial attention, its impact is expected to be considerable. By incorporating the findings of this study, many machine learning models could also benefit from our results.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper is limited to technical content and does not inherit the risks. The model and benchmark data do not appear to carry significant misuse risks warranting special release restrictions beyond standard open-source protocols.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.

- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper appropriately cites sources for existing assets such as baseline methods (e.g., S4, S4D, Mamba, MF, IPC) and datasets (e.g., LRA, FineWeb). We include the corresponding source links and license information in the appendix. All assets are used in accordance with their respective licenses.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: This study proposes a theoretical method aimed at improving the performance of machine learning models and does not involve the creation of new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The study does not include any experiments involving crowdsourcing or human participants.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: IRB approval is not applicable to this work, as it does not involve research with human participants.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs were used solely for proofreading purposes, such as correcting grammar and improving the naturalness of the language for non-native English authors. They did not contribute to the core research ideas or findings.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.